1988

System identification of the smoke opacity vs fuel metering in a diesel engine

Han-Keun Cho
Iowa State University

Order Number 8909135

System identification of the smoke opacity vs. fuel metering in a diesel engine

Cho, Han-Keun, Ph.D.

Iowa State University, 1988

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

System identification of the smoke opacity vs. fuel metering in a diesel engine

by

Han-Keun Cho

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Agricultural Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1988

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Problem and Background

For the past decade, the energy crisis has forced rapid changes in internal combustion engine design and operating practices to maximize fuel economy. Also, finding alternative fuels for whole or part substitution of fossil fuel has been a major research topic. Public interest has motivated intensive research and development for reducing emissions from internal combustion engines. Thus, the development of new internal combustion engines and control devices to maintain optimum performance while reducing emissions has been regarded as an important concern by the mobile-power industry.

Because of the superior fuel economy (Black and Scahill, 1983) and the potential applicability to synthesized fuel, the use of high-speed direct injected diesel engines has been continuously increasing in agricultural and industrial applications. Although the diesel has lower level of carbon monoxide and unburned hydrocarbons, unfavorable particulate emissions may discourage wide-spread use (Wade, 1981, Alkidas, 1984). Particulate emissions normally appear as black smoke and not only cause health hazards but limit maximum power output. Generally, the level of black smoke is not severe in properly tuned diesel engines running under steady state conditions. Smoke problems generally appear during unsteady operation (Henein, 1979). The change of fuel metering rate caused by rapid governor response to either speed

change or load change often results in increased smoke when normal mechanical fuel injection systems are used.

Control methods for exhaust emission can be divided into two categories: new hardware design and optimum engine control system design. It is impossible to achieve optimum levels of performance without combining both methods. For a given engine, however, controlling operating variables is still the most efficient way to reduce exhaust system smoke. Engine modifications, such as exhaust gas recirculation, air swirl and high pressure fuel injection, and attachment of after-treatment devices on the exhaust system, such as particulate filters and traps, are known to be effective ways to reduce particulate levels. Practical implementation requires more investigation and improvement. It has become obvious that new versions of internal combustion engines will include micro-computer based fuel injection systems which include emission control features in addition to precise speed control characteristics.

The design of engine control systems requires information regarding the dynamic and nonlinear engine characteristics. These include a complicated combustion process. Extensive study has been continued by many researchers to reduce exhaust emission levels by improving the combustion process through better understanding of the fundamental mechanisms, but the details of the emission formation processes have not been fully recognized. Some mathematical models of diesel engines have been reported (Shroff and Hodgetts, 1974, Shahed et

al., 1975 and Dent and Mehta, 1981); however, those models based on physical interactions are not sufficiently developed for application in a diesel engine control system. Therefore, current engine control system development is almost always based on empirical models derived from measured engine data for a given engine and specific test conditions. Because of the inherent complexity of the diesel combustion process and the consequent difficulty of developing useful dynamic models, control design for diesel engines tends to utilize a simpler method. For the last decade, gasoline engine control systems based on empirical models have been developed by Fruchete and Kate (1978), Cassidy (1978) and Chang and Sell (1983). While this approach has the disadvantages in applicability to other systems, it showed substantial improvement over the actual engine data approach. Zhang et al. (1985) developed a tractor diesel engine control system using a dynamometer mapping method to improve fuel economy without considering exhaust emission.

Changes in diesel engine operation due to changes in actual operation, such as speed and load, cause parametric errors in the control system. The maintenance of performance over the lifetime of a diesel engine is one of the most difficult problems since changes in component characteristics due to aging, corrosion, or other factors can not be accomplished with a standard conventional controller (Sweet, 1982). A general approach to the problem of severe dynamic changes is the use of an adaptive control system, such as adaptive diesel speed

control by Wellstead (1981) and the knock adaptive improvement in spark ignition engine by Sawada and Shigematsu (1981). Adaptive control systems are generally defined to include all types of control strategies that are modified to account for parametric variations in the system, preserving uniformity of overall system characteristics. The adaptive control process generally performs three functions: identification, decision and actuation. For adaptive control design, the identification should be performed in a short time period to track process changes so that the adaptive controller can maintain system performance. A proper identification method, which provides the information required to design or adjust the controller, must be simple and conveniently implemented.

Objectives

Reduction of diesel smoke by computer control was the ultimate goal of this research. An adaptive control scheme may be implemented for developing a new fuel injection system to control the engine speed within the allowable exhaust smoke level. The focus of this study was to determine the effect of fuel metering on the smoke emission. As a first approach in developing an adaptive-control fuel injection system, the identification of the system transfer function between fuel metering rate and smoke exhaust level must be determined to obtain a base model.

Specific objectives were;

1. To develop a personal computer based identification package for use in diesel engine modeling,
2. To test system identification methods with a simulated process, and
3. To experimentally determine the transfer function between fuel metering rate and smoke exhaust level for a specific diesel engine.

LITERATURE REVIEW

System Identification

During the past couple of decades, many different identification and parameter estimation methods for dynamic processes have been developed and reported in the literature. It is very difficult to get an overview of a field in rapid development, but it is possible to point out a few important aspects for understanding the subject.

Definition and procedure

Zadeh (1962) defined the classical meaning of identification: "Identification is the determination, on the basis of input and output, of a system within a specified class of systems, to which the system under test is equivalent." Eykhoff (1981) explained this definition as: 1) the use of 'a priori' knowledge, for the set of systems or models based on the laws of physics, as well as 2) the use of observations for the necessary 'a posteriori' knowledge on the system. Ljung (1987) discussed three basic entities about the system identification procedure: 1) the input-output data recorded during a specifically designed identification experiment, 2) a set of candidate models with 'a priori' knowledge intuition and insight and 3) determining the best model in the set. A schematic of identification loop as in Figure 1 shows that the procedure would repeat until the obtained model passed the validation test.

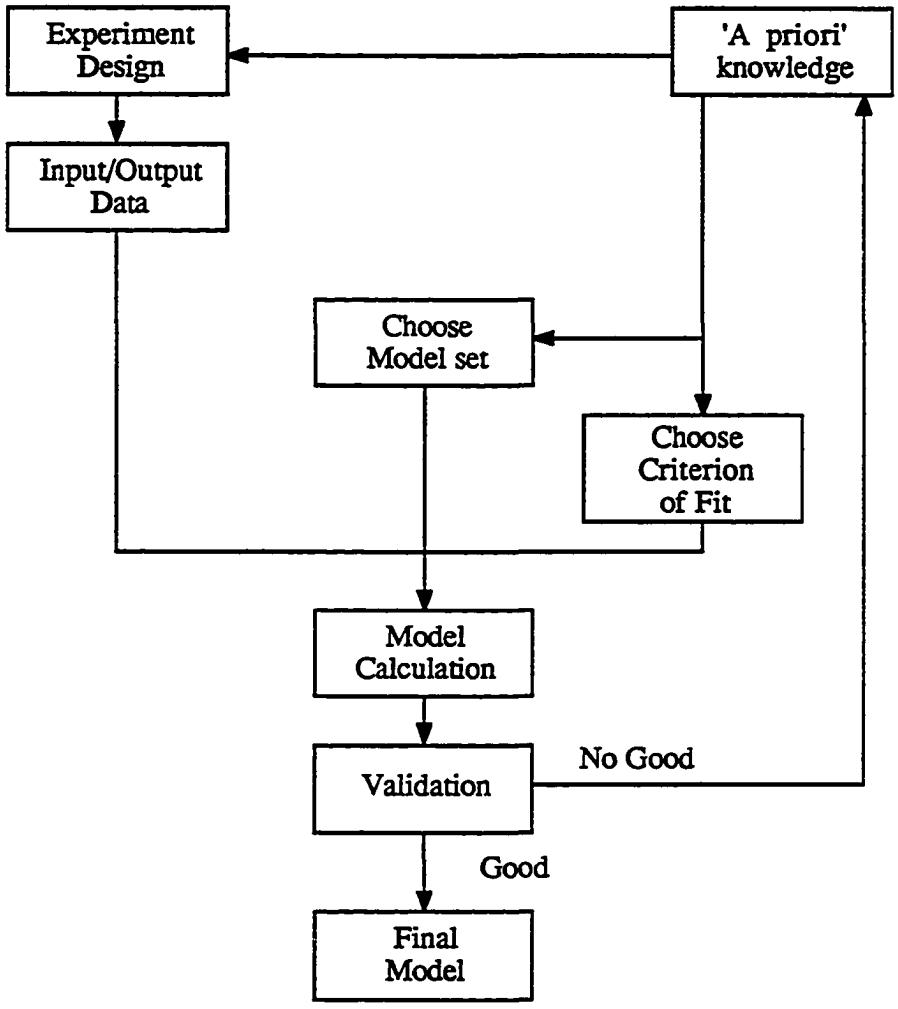


FIGURE 1. The system identification loop (Ljung, 1987)

Classification of identification method

Identification methods can be classified according to the basic elements of the problem, i.e., the class of model, the input signal, application and data processing. Classification has been done extensively in Balakrishnan and Peterka (1969), Astrom and Eykhoff (1971), Nieman et al. (1971), Gustavsson (1975), Sawaragi et al. (1981) and Ljung (1987).

Class of models The models can be characterized by 1) nonparametric representations such as impulse response in the time domain, transfer functions in the frequency domain, covariance functions and spectral densities

$$T_{ij}(s) = \frac{Y_i(s)}{U_j(s)} \qquad \begin{array}{l} U_k(s) = 0,\ k \neq j \\ U_k(0) = 0, \text{ for all } k \end{array}$$

where $T_{ij}(s)$ is a matrix of transfer functions, $Y_i(s)$ is a transformed output vector, $U_j(s)$ is a transformed input vector, and 2) parametric representation, such as state models

$$\frac{dx_{ij}}{dt} = f(x_{ij}, u_j, \beta_k)$$

$$y_i = g_i(x_{ij}, u_j, \beta_k)$$

where x_{ij} is the state matrix, u_j is the input vector, y_i is the output vector and β_k is a parameter vector. Nonparametric methods do not explicitly require a finite-dimensional parameter vector for determining transfer functions by direct techniques. Parametric methods, however, require assumptions to be made about the choice of a set of candidate dynamic models. A parameter vector is used for a model structure and a search is made for the best model that estimates the parameter vector. Ljung and Glover (1981) compared the frequency domain method with the time domain method and discussed technical differences between the approaches. They concluded that these are complementary rather than competing techniques. A general tutorial survey of nonparameteric methods was given in Wellstead (1981) and theoretical results were well covered in Ljung and Glover (1981). Parameter estimation methods are typically explained in the field of statistics (Cramer, 1946, Rao, 1973 and Lindgren, 1976).

Class of input signals Identification methods can be classified according to the type of input signals, i.e., deterministic or stochastic. In the case of a deterministic method, signals such as pulse, step, sinusoidal and binary signal (pseudo-random, multi-frequency and optimal) are used as inputs to the system. In the case of the stochastic method, noise or a random signal, such as colored or white noise, are used to perturb the input to perform the identification.

Classical and modern methods The classical method which means "tried and proven to be simple and yet powerful" generally determines the impulse response or transfer function of a system. Such methods are frequency analysis, transient analysis and correlation methods. Sawaragi et al. (1981), Eykhoff (1974), Sage and Melsa (1972), Davies (1970), Chen and Hass (1968) describe theories and main arguments about these techniques. The identification methods which do not belong to the classical methods have mostly developed in recent years and classified as modern methods. These methods are least square methods, maximum likelihood methods, instrumental-variable methods, etc. Those methods are broadly discussed in Hsia (1977), Akaike (1981), Rissanen (1985), and Ljung (1987).

Application

Gustavsson (1975) extensively surveyed the application of system identification techniques to chemical and physical processes. He found that the most popular methods adopted in practical applications were correlation analysis, step and frequency response techniques, and the least-squares estimation method, since all these methods were simple to use, cheap to implement and most often gave satisfactory results. His survey concludes with showing that identification is a very useful tool to obtain models in process industries without too much cost and trouble, although there are still great difficulties in modeling complex processes.

A series of frequency response tests was performed to determine the dynamic behavior of diesel engine for control purposes (Bowns, 1970, Hazell et al., 1971). These experiments used commercially available, cross-correlation-type transfer function analyzers (TFA). Some years later, Windett and Flower (1974) included a synchronization scheme to eliminate the scatter experienced by previous researchers. The scatter was explained as a function of the measurement in their work.

Morris et al. (1982) implemented an identification process to develop a spark ignition engine model for computer control purposes. They determined the digital transfer function relating output torque with throttle position using the model reference identification technique.

Backhouse and Winterbone (1986) studied the transient behavior of torque and smoke from a turbocharged diesel engine by the frequency response method. The frequency response measurement was obtained by the correlation type of commercial analyzers using a pseudo-random binary sequence as an input signal.

Experimental design

The subject of informative experimental design has received considerable attention in the field of system identification, since changing to a new experiment after an initial failure could be a costly and time-consuming procedure. Goodwin and Payne (1977) describe the typical constraints on the allowable experimental conditions which must

be considered in experimental design; 1) amplitude and power constraints on input/outputs variables, 2) total number of samples and sampling rate, and 3) availability of hardware and software. Ljung (1987), Goodwin (1987) and Gustavsson (1975) also reported comprehensive studies on this subject.

Choice of variables The first step is deciding which signals are to be considered as inputs or outputs. The inputs are variables to be set and the outputs are variables to be measured. If the model is to be developed for a controller design, the inputs and outputs would be matched with those for the controller.

Input signal The choice of input signals has been considered one of the most important factors in experimental design because of the substantial influence of the input signals on the observed data and their fundamental role in determining the nature and accuracy of the system characteristics. Step or sinusoidal inputs are the simplest and oldest signals used in identification. The fact that the system must be significantly disturbed to obtain useful information, however, has prevented their wide application. Although pulse input is reasonably simple, takes less test time and gives more accurate results than step or sine wave inputs, the practical implementation of signal characteristics is difficult to achieve without violating linearity constraints by increasing pulse height while decreasing width. Casset and Mellichamp (1975) employed an improved multi-frequency input signal built from a set of sine waves with appropriate frequencies to reduce

identification time without violating linearity constraints. It was noted that the amplitude of the composite sinusoidal input signal must be still higher than a well-designed binary sequence with approximately equal Fourier coefficients at the same set of frequencies. Binary signals, such as pseudo-random binary sequences (Briggs et al., 1967), have been used extensively in many applications since step or sinusoidal inputs lead to long measurement times and have a large noise level (Gustavsson, 1975). Godfrey (1970) gives a survey of pseudo-random binary sequence applications. Cumming (1970) showed that binary inputs of the synchronous random telegraph wave performed better than pseudo-random binary sequency when matched to the band-width of the system. Van den Bos (1967) developed an algorithm to design multi-frequency binary signals that have the major part of their power concentrated in predefined frequencies. Harris and Mellichamp (1980) applied a multi-frequency binary signal on a simulated system and on two bench-scale systems, one having a significant delay time and the other having significant nonlinear characteristics. Their results showed that their model could be used as an expression of an actual physical system in a low noise situation and as a control model in a high noise situation.

Sampling interval It is important to select the sampling rate so that information loss, which is unavoidable in data-acquisition, is insignificant. Gustavsson (1975) suggested performing several experiments with different sampling rates and input signal

characteristics in order to cover the whole frequency range of interest. As a practical rule, he recommended using a sampling interval of the smallest time constant or using a sampling interval of about one-tenth of the major time constant. Finally, it was advised to sample as fast as practically possible then to analyze the data only every other or so datum if it turned out that this was sufficient. More recently a few theoretical results have been reported by Walberg and Ljung (1986) and Ljung (1987). These reference discussed the choice of the sampling interval regarding aliasing effects, bias considerations and variance considerations, and concluded that: 1) very fast sampling causes calculation problems, model fits in high-frequency bands, and poor returns for extra work; 2) slower sampling than the natural time constants leads to drastic increase of the variance; 3) optimal choices of sampling rate for a fixed number of samples lie in the range of the time constants of the system; and 4) a sampling frequency that is about ten times the bandwidth of the system should be a good choice in most cases.

Diesel Exhaust Smoke

Diesel smoke production

Diesel smoke is defined as "particles including aerosols, suspended in the engine's gaseous exhaust stream which obscure, reflect and/or refract light" (SAE J255a, 1978). The particulates from a diesel engine can be divided into 1) soot or black smoke as a product

of the incomplete combustion or abnormal engine operation, 2) liquid particulates as white/blue clouds of vapor mainly consisting of unburned fuel, a lubricant and its partial oxidation products, and 3) other particulates resulting from lubricants and additives (Henein, 1979).

To understand the formation process of the particulates as well as other exhaust emissions, a detailed understanding of the diesel combustion process is extremely important. Diesel smoke formation is a very complex process involving complicated and variable geometry configuration, and various temperatures, pressures and air/fuel ratios taking place in extremely short period of time (i.e., 1 - 10 ms). Because of the complexity of the diesel combustion process and difficulty in instrumenting the diesel cylinder to investigate the particulate formation process, much of the initial work on diesel particulate research has been limited to characterization of particle size and composition (Vuk et al., 1976, Kittleson et al., 1978, Dolan and Kittleson, 1978, and Groblicki et al., 1979).

A general description for diesel combustion process, which will be helpful in understanding the smoke formation in diesel engines has been introduced by some previous researchers (Yu et al., 1982, Henein, 1979, and Van Gerpen, 1984). Diesel combustion can be divided into three periods: ignition delay, premixed combustion and mixing controlled combustion. The time between the start of injection and autoignition is the ignition delay, during which the injected fuel forms droplets,

evaporates and mixes with air to form a combustible mixture (physical delay), and the chemical reaction produces preignition radicals for rapid combustion (chemical delay). Ignition delay has been found to be affected by the air temperature, pressure and oxygen concentration, all of which influence the rate of chemical reactions. Ignition starts at multiple nuclei developed previously causing a rapid pressure and temperature increase in the cylinder and continues until all of the fuel-air mixture that has a fuel-air ratio close to stoichiometric burns out. Since the remaining fuel in the spray core is too rich to burn, this fuel burns much more slowly at a rate controlled by air-fuel mixing. This mixing controlled combustion period contributes to providing a good environment for pollutant formation such as soot and NO_x. Although the overall fuel-air ratio is always lean in a diesel engine, rich zones exit in the spray core which mix with hot products and unused air according to the amount of turbulence in the cylinder. If mixing is not rapid enough, these rich regions can reach the high temperatures necessary for gas phase pyrolysis and dehydrogenation of the fuel molecules, resulting in soot production. After mixing with air, the soot oxidation process occurs at a rate that has kinetics controlled by the local oxygen concentration and the temperature.

To help in understanding the basic principle of diesel particulate production, Amann et al. (1980) collected the existing sources of knowledge outside the engine environment, such as chemical equilibrium, laboratory test results, fundamental studies of smoke and industry

reports. They considered three aspects of the particulate formation process; 1) combustion environment responsible for soot formation, 2) the formation mechanism, and 3) the oxidation process. They pointed out that: 1) diesel combustion is a diffusion controlled process and soot formation is a kinetically controlled; 2) it is advisable to ensure locally high equivalence ratios, to use fuels with high hydrogen-carbon ratios, and to maintain high reaction temperature in order to reduce the thermochemical potential for soot formation; 3) soot production starts with particle nucleation, followed by growth through both surface deposition and coagulation which, in diesel cylinder, is essentially completed before the exhaust valve opens; and 4) increased oxygen partial pressure caused by mixing air into the packet favors soot oxidation, however, that mixing lowers packet temperature which retards oxidation. Soot formed and mixed during combustion period is likely to oxidize easily.

Hiroyasu et al. (1980) performed a set of combustion experiments to obtain a fundamental understanding of soot formation and oxidation during the combustion period in diesel engines. As a first experiment, a flame luminosity detector with a phototransistor was implemented to measure the combustion duration (or soot combustion) in a cylinder. It was noticed that soot was formed mostly from the early to middle stages of combustion and burned mostly from middle to final stages of combustion. Thus, exhaust smoke level will be lower when the whole combustion process ends before the exhaust valve opens. This can be

accomplished by advancing injection timing or by adding air swirl. As a second experiment, a continuous spray diffusion flame simulating diesel combustion was used and the concentration of soot was measured by a nitrogen gas quenching probe. The results showed that the location of maximum soot did not correspond to that of maximum temperature, but to that of minimum oxygen concentration. In the last experiment, it was shown that the air jet could be used to control the flame length in the spray open flame and that the air swirl is an effective method of controlling combustion duration in the cylinder. The increase in the air-fuel mixing rate by the combination of air swirl and fuel injection parameters, however, promotes rapid burning and results in a larger portion of the fuel mixture burning near top dead center, causing cylinder pressure and gas temperature to rapidly increase to high levels, which increase NO_x production and noise level. With this accelerated burning, the combustion process approaches the thermodynamically ideal constant volume process, which increases engine efficiency. Thus, a study of engine performance and emission includes those variables and their relationship (Van Gerpen, 1984).

Norris-Jones et al. (1985) investigated the formation of particulates in the cylinder of a swirling direct injection diesel engine using high-speed combustion photographs and in-cylinder sampling techniques. Their results showed that the level of particulates formed from the fuel burning of droplets and 'off-the-wall' conditions is higher than that formed from the fully evaporated conditions. They as

well as others have reported that smoke and particulates decrease with injection timing advance in a direct injection engine.

Factors affecting diesel smoke

The level of particulate emission produced by a diesel engine depends on the engine design, the operation mode, the properties of the fuel used and the ambient conditions. The effects of these variables on particulate emission have been studied by many researchers (Henein, 1979, Hames et al., 1971, Khan et al., 1972, Bryzik and Smith, 1978, Van Gerpen, 1984). Those variables include fuel type, injection system (rate, timing, and nozzle configuration), inlet air temperature, air swirl level, air-fuel ratio, and engine speed.

Fuel The fuels having a higher cetane number are known to produce more smoke due to lower stability of these fuels. The effects of fuel volatility on smoke production, however, is not well known (Henein, 1979).

Injection system The size of nozzle orifice and the ratio of orifice length to diameter affect smoke production. A larger hole size results in less atomization and increased smoke (Hames et al., 1971). An increase in the ratio of hole length to its diameter beyond a certain limit also results in increased smoke. If the total flow area is constant, a larger number of holes results in increased smoke due to less swirl (Lyn, 1970).

Higher injection rate tends to reduce smoke and increase NO_x emissions (Greeves, 1979, Hames et al., 1971). Greeves, however,

reported a maximum injection rate after which no further reduction in smoke was observed.

Advanced injection timing results in more fuel injected before ignition due to longer delay periods, earlier ending of the combustion, and increased residence time. These factors are effective in reducing the smoke level. Earlier injection, however, causes more noise, higher mechanical and thermal stresses, and higher NO_x emissions (Khan et al., 1972).

Inlet air temperature An increase in inlet air temperature generally results in increased smoke intensity. Henein (1979) explains this as; the higher gas temperature during the entire cycle due to higher inlet air temperature affects both the spray characteristics, such as lower penetration rate and higher rate of evaporation and diffusion, and chemical reaction, such as faster decomposition reaction. Those effects result in locally rich fuel droplets near the nozzle and increase the smoke level.

Air swirl level Increasing air swirl results in higher temperatures and lower equivalence ratios in combustion zones causes decreases in smoke level and increases in NO_x emissions (Wilson et al., 1974, Khan et al., 1972). Van Gerpen et al. (1986) observed that particulate emissions increased due to higher swirl in small hole nozzles. Veselinovic (1985) states that swirl level in swirl chamber determines the amount of diffusion burning which directly affects the smoke level.

Air-fuel ratio Air-fuel ratio is known to affect temperature and partial pressure of fuel in pyrolytic reaction and flame, influencing the smoke production. This factor affects not only duration of fuel injection resulting in the change of fuel injected after combustion but fuel distribution regions in the spray resulting in the structure change of smoke formation (Henein, 1972). At heavy loads where the air-fuel ratio is richer than 20:1, smoke in the exhaust increased significantly (Wade, 1981).

Speed Engine speed mainly affects the degree of turbulence within the cylinder and the time available for mixing and chemical reaction. The effect of turbulence on combustion is not well known except those effects on both mixing of air and fuel, and flame temperature. The effect of time available is, however, obvious; at higher engine speeds, the carbon formed during the combustion process has less time to mix and react with oxygen resulting in higher smoke levels (Bryzik and Smith, 1978).

Smoke opacity measurement

An accurate and consistent measurement of diesel smoke is important in research, development, certification and production testing. The steady-state measurement of visible, black smoke is well described in the SAE information report (SAE J255a, 1978). The direct measurement of smoke opacity can be divided into two methods; 1) the filtering method and 2) the light extinction method. The filtering method involves measuring of the amount of soot particles collected on

a filter material passed by exhaust gases. Various methods can be used to evaluate the filter; comparison with standards, measurement of photoelectric reflectance and determination of the sampled mass. Light extinction methods can be classified with full flow types (in-line and end-of line) and sampling types.

To express the relation between the opacity of a smoke plume, the effective optical path through the plume and the opacity of the smoke per unit length, the Beer-Lambert equation can be used:

$$N = 100 \{ 1 - e^{-KL} \}$$

where N = opacity, %

e - base of natural logarithms

K = absorption coefficient or smoke density, m^{-1}

L = effective path length through the smoke, m

$$K = n A Q$$

where n = number of particles per unit volume, particles/m^3

A = average projected area of each particle, m^2/particle

Q = specific absorbance

This equation was validated in a test conducted by the SAE Diesel Smoke Measurement Task Force (SAE J255a, 1978). For accurate measurement of transient smoke concentrations, full-flow light extinction type smoke

meters are generally employed, since sampling type light-extinction meters have design limitations. The calibration procedures for instruments of this type are described in SAE Recommended Practice (SAE J1157, 1976).

IDENTIFICATION METHOD

Model

Diesel engine operation can be assumed to be linear in a limited region about the normal operating state, thus, linear theory was adopted for identification, although the diesel combustion process is nonlinear when operating over its entire range of speed (Backhouse and Winterbone, 1986). Based on the time-invariance effect of a linear model, frequency domain analysis was applied. The smoke production in diesel combustion is caused by many design and operating parameters, such as air/fuel ratio, injection system and engine speed (Bryzik and Smith, 1978). The level of black smoke is not significant in steady state operation with properly operated diesel engines (Henein, 1979) and the fast change in fuel metering rate caused by poor control of fuel injection pump increases the smoke exhaust level. Thus, fuel metering rate is one of the most important operating variables in smoke control studies. Thus, a single input (fuel rate) and single output (smoke intensity) model was selected. A general parameter model for single input/single output can be described by the differential equation,

$$a_n\, y^{(n)}(t) + \ldots. + a_0\, y^{(0)}(t) = b_0\, x^{(0)}(t-\tau_d) + \ldots. + b_m\, u^{(m)}(t-\tau_d) \qquad (1)$$

or by the corresponding frequency response

$$H(j\omega) = \frac{b_0 + \ldots\ldots + b_m(j\omega)^m}{a_0 + \ldots\ldots + b_n(j\omega)^n} \exp(-\tau_d j\omega) \qquad (2)$$

where $y^{(n)}(t)$ is n th derivative of measured output with respect to time, $x^{(m)}$ is the m th derivative of input signal with respect to time, a_n and b_n are parameters and τ_d is dead time. In the time-invariant linear operations, the operation is governed by the convolution integral.

$$y(t) = \int_{-\infty}^{+\infty} h(\tau)\, x(t-\tau)\, d\tau \qquad (3)$$

where x(t) denotes the input signal; h(t) is the characteristic signal identifying the operation process; and y(t) is the corresponding response signal. Transforming this response signal in the frequency domain will result in

$$Y(s) = H(s)\, X(s) \qquad (4)$$

where Y(s), H(s) and X(s) are the Laplace transforms of y(t), h(t) and x(t) respectively. The transfer function H(s) of a system with excitation x(t) and response y(t) is defined as the ratio of transformed output to input signal under the condition of zero initial stored energy in the system (Cadzow and Landingham, 1985).

$$H(s) = \frac{Y(s)}{X(s)} \quad \text{at zero initial energy in the system} \tag{5}$$

To determine the frequency response in the steady state by Fourier transform, replacing s with $j\omega$ will result in

$$H(j\omega) = \frac{Y(j\omega)}{X(j\omega)} \tag{6}$$

where ω = frequency

Input Signal

Traditional identification procedures subjecting the system to step, ramp, pulse or sinusoidal input variations are simple to use and easy to analyze. The constant frequency response techniques may provide more dynamic information than the others. This method however requires long experimental times (Cusset and Mellichamp, 1975). Also they are not always practical to use because the system must be significantly disturbed to minimize the noise effect for the accurate dynamic information. For this reason, many researchers have recommended the use of a well-designed binary sequence; these may be pseudorandom binary sequences, optimal binary sequences and binary

multi-frequency sequences. Binary sequences, such as pseudo-random binary sequences (PRBS) which have a uniform energy spectrum have been used extensively in many applications. More recently multi-frequency binary sequences were designed to concentrate their energy at the discrete frequencies of interest (Van den Bos, 1967) and showed improvements over the pseudo-random binary signal when the test sequence frequency bandwidth is similar to the system bandwidth (Cumming, 1970, Van den Bos, 1973). In this study, PRBS and multi-frequency binary signal were used as input signals.

Pseudo-random binary signal

Properties of PRBS The signal has two levels of state and switches from one level to the other only at discrete intervals of time. The change occurs in a deterministic pseudo-random manner, and the sequence is periodic, PRBS is thus deterministic and periodic, and experiments are repeatable. These signals all satisfy a set of conditions of randomness: 1) balance property, 2) run property, and 3) correlation property (Schwarzenbach and Gill, 1984).

Generation procedure The most commonly-used PRBS is based on maximum-length sequences (m-sequences), for which $N = 2^n - 1$ and where n is an integer. These may be generated using an n-stage feedback shift register, with feedback to the first stage consisting of the modulo-2 sum of the logic level of the last stage and one or more of the other stages. Very few feedback connections in fact yield a maximum-length sequence for any particular n; Table 1 lists some

appropriate connections for $2 \leq n \leq 11$. A shift-register circuit for generating a PRBS of length 127 is shown in Figure 2. Figure 3 shows the PRBS of length 127 used in this study and Figures 4 and 5 show its auto correlation function and power spectral density respectively. A sequence generating program is described in Appendix D.

TABLE 1. Suitable feedback connections for m-sequence generation

No. of shift register stage, n	Periods of sequence, $N = 2^n - 1$	Feedback to first stage modulo-2 sum of signal
2	3	1 and 2
3	7	2 and 3
4	15	3 and 4
5	31	3 and 5
6	63	5 and 6
7	127	4 and 7
8	255	2, 3, 4, and 8
9	511	5 and 9
10	1023	7 and 10
11	2047	9 and 11

Binary multi-frequency signal

Criterion The multi-frequency binary signals having most of their power distributed in a selected number of frequencies were generated by the method reported by Van den Bos (1967). The procedure combined a systematic and random search which does not necessarily yield the optimal solution. A sufficiently good suboptimal signal, however can be found through replication. The criterion I, which must be minimized, is defined as

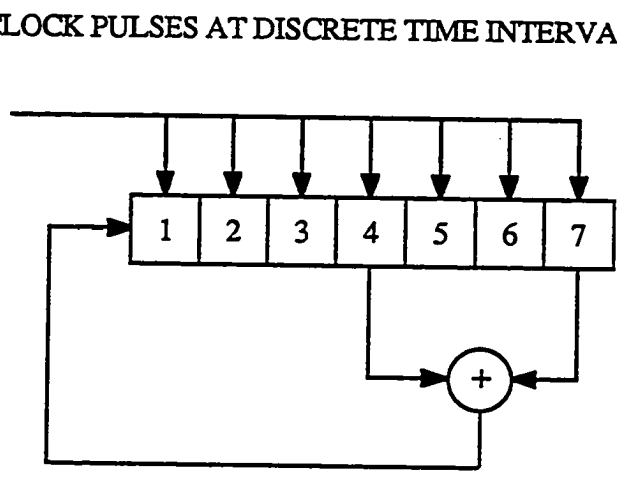


FIGURE 2. Shift register circuit for generating a 127-digit m-sequence

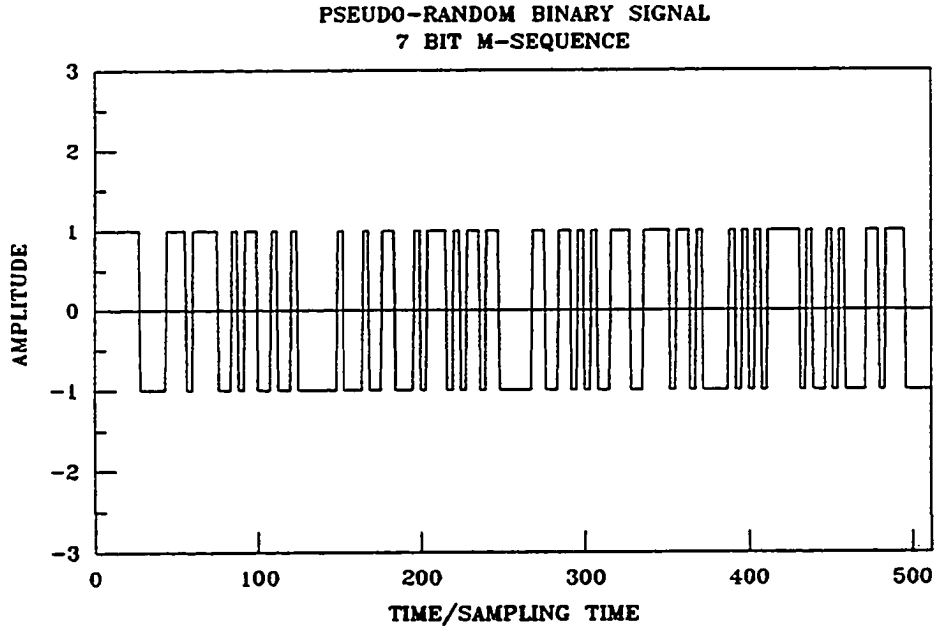


FIGURE 3. Pseudo-random signal used in this study

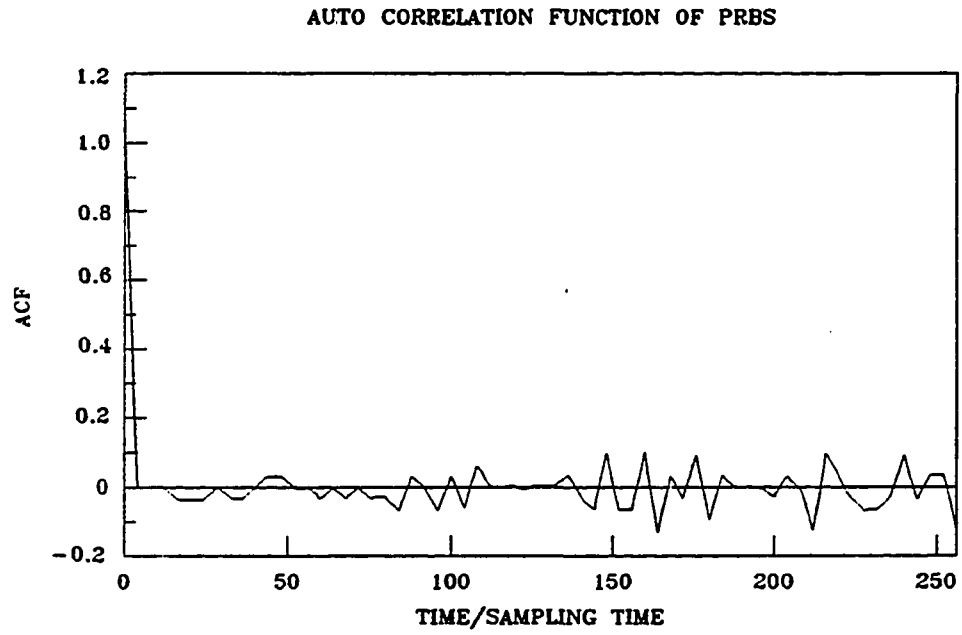


FIGURE 4. Auto correlation function of pseudo-random signal

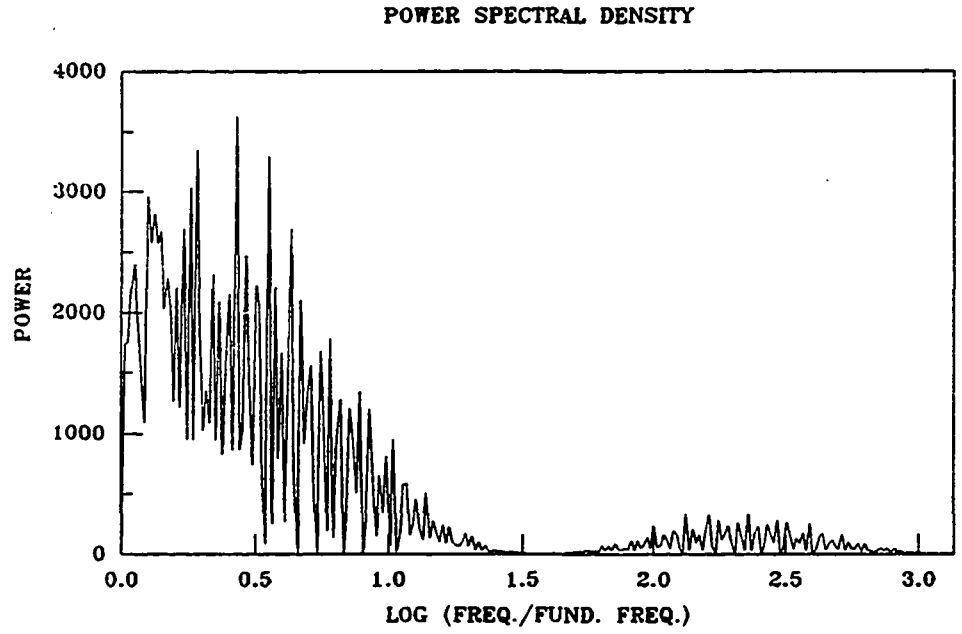


FIGURE 5. Power spectral density of pseudo random signal

$$I = \sum_{i=1}^{N} \{ P(\text{ideal})_i - P(\text{actual})_i \}^2 \qquad (7)$$

where $P(\ )_i$ = Power at the selected frequency

N = Number of the selected frequencies

The total power contained in a periodic signal u(t), P_{total} is defined as

$$P_{total} = \frac{1}{2T} \int_0^T u^2(t)\, dt \qquad (8)$$

where u(t) = ±1 for a binary signal

thus, P_{total} becomes 1. Since as much as power as possible must be distributed uniformly over the N selected frequencies, $P(\text{ideal})_i$ will be as 1/N. The actual power at the frequency i can be defined as

$$P(\text{actual})_i = 2\, |F(j\omega_i)|^2 \qquad (9)$$

Thus, the criterion I will be changed to

$$I = \sum_{i=1}^{N} \{ 1/N - 2|F(j\omega_i)|^2 \}^2 \qquad (10)$$

Procedure Since the set of signals to be generated is an even function of time, the first half of the fundamental period is considered. One set of random binary signals is generated initially for the first half period and the criterion I in equation (10) is calculated. Then the signal on a randomly selected interval is switched. If the change is an improvement in the criterion value it is kept, if not, the original signal is restored. Then the same is done with the remaining intervals. A new initial interval with a new starting point is selected and process is continued until a complete cycle without further improvement occurs. The whole procedure is repeated a number of times and the best resulting signals are selected. The best power distribution at the specified frequencies (1,2,3,4,6,8,12 and 16 times the fundamental frequencies) was 75%. Figure 6 shows the binary signal generated using this procedure. The magnitude of the signal in the frequency domain shown in Figure 7 tells the degree of power distribution at the frequencies of interest. A sequence generating program is described in Appendix D.

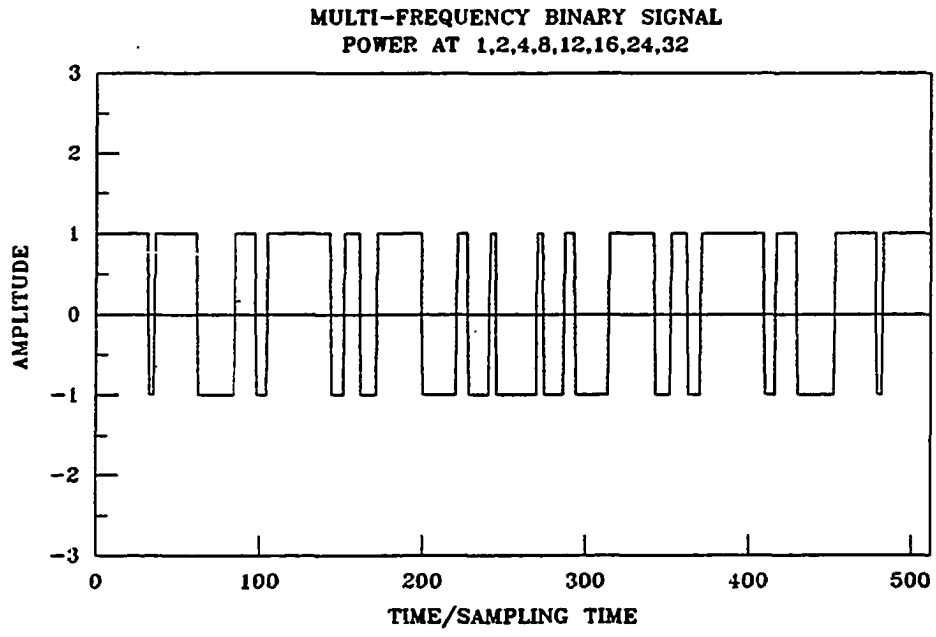


FIGURE 6. A binary multifrequency signal in time domain

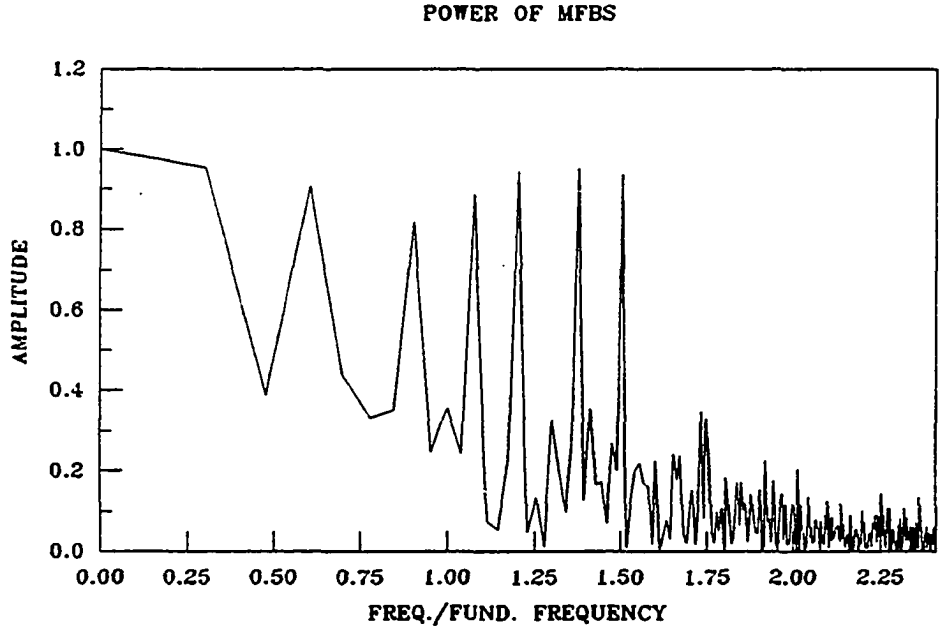


FIGURE 7. Magnitude of binary signal in frequency domain

Mathematical Tools

Fast Fourier transform

The Fourier transform of x(t) is defined as

$$X(f) = \int_{-\infty}^{+\infty} x(t)\, e^{-j2\pi ft}\, dt \qquad (11)$$

To transform a time series of samples to a series of frequency-domain samples, the discrete Fourier transform (DFT) can be derived by several modifications of equation (11).

$$X_d(k) = \sum_{n=0}^{N-1} x(n)\, \exp\{(-j2\pi n/N)k\} \qquad (12)$$

$$x(n) = \sum_{n=0}^{N-1} X_d(k)\, \exp\{(-j2\pi k/N)n\} \qquad (13)$$

where N = number of samples

n = time sample index, n=0,1,2,..,N-1

k = discrete frequency component index, k=0,1,2,..,N-1

Cooley and Tukey (1965) reported a faster algorithm for numerical calculation of the Fourier transform, known as the fast Fourier transform (FFT). The FFT reduces the number of operations and for

large N, the time advantage is tremendous ($N \log_2 N$ vs. N^2). In this study, the FFT algorithm was implemented to transform the input/output signals forward or backward.

Spectral analysis

The Fourier transform of the auto-correlation function defined by

$$\Phi_{xx}(j\omega) = \int_{-\infty}^{+\infty} \phi_{xx}(\tau) \exp(-j\omega\tau)d\tau \qquad (14)$$

is known as the power density spectrum or power spectral density. The cross-spectral density function between two signals x(t) and y(t) is similarly defined:

$$\Phi_{xy}(j\omega) = \int_{-\infty}^{+\infty} \phi_{xy}(\tau) \exp(-j\omega\tau)d\tau \qquad (15)$$

If the output of the system is corrupted with noise n(t) which includes both noise introduced by the measuring device and noise due to other sources so that the Equation (3) can be modified to

$$z(t) = \int_{-\infty}^{+\infty} h(\tau)\, x(t-\tau)\, d\tau + n(t) \qquad (16)$$

With several modifications, this equation becomes

$$\phi_{xz}(t) = \int_{-\infty}^{+\infty} h(\tau)\phi_{xx}(t-\tau)\, d\tau + \phi_{xn}(t) \qquad (17)$$

If x(t) and n(t) are uncorrelated, $\phi_{xn}(t)$ becomes zero as the correlation time is increased.

$$\phi_{xz}(t) = \int_{-\infty}^{+\infty} h(\tau)\phi_{xx}(t-\tau)\, d\tau \qquad (18)$$

With Fourier transform, the frequency domain relation

$$\Phi_{xz}(j\omega) = H(j\omega)\, \Phi_{xx}(j\omega) \qquad (19)$$

and

$$H(j\omega) = \frac{\Phi_{xz}(j\omega)}{\Phi_{xx}(j\omega)} \qquad (20)$$

can be obtained.

Optimization

To fit the nonparametric results, which are complex values of the transfer functions, to a parametric model, the complex method of Box (1965) was implemented. This complex search algorithm has proved

effective in solving problems with nonlinear objective functions subject to nonlinear inequality constraints. This algorithm starts with a number of initial sets (complex) of points which are feasible starting points and additional points. These sets of initial points are tested against the constraints and then changed a small distance inside the violated limit if they are out of bounds. At each point, the objective function is calculated and the point having the highest function value is replaced by another point that is reflected through the centroid of remaining points. If the point having the highest function value repeats itself, it is moved one half the distance to the centroid of the remaining points. This process is repeated until the object function value converges a predefined value (Equation 21). The flow diagram shown in Figure 8 illustrates this search procedure.

$$F_{obj} = \sum_{i=1}^{N_f} \{ T(cal)_i - T(pos)_i \}^2 \qquad (21)$$

where F_{obj} = Object function

$T(\ )_i$ = Calculated or postulated transfer function at the selected frequency

N_f = Number of the selected frequencies

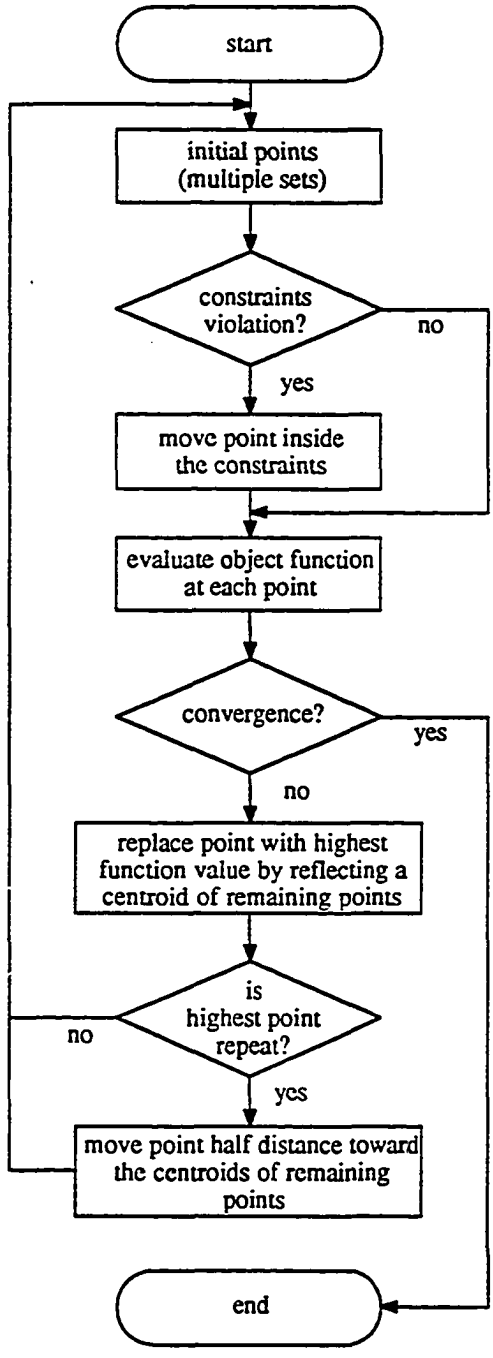


FIGURE 8. Flow chart of the optimization procedure for parameter search

Simulation Study

Process simulation

Frequency response of model equation described in Equation (2) can be simplified as follows:

$$\frac{Y(s)}{X(s)} = \frac{K_p\, e^{-Ds}}{(\tau_1 s + 1) \cdots (\tau_n s + 1)} \qquad (22)$$

where K_p = process gain

D = dead time

τ_n = time constant, n = 2,3 and 4

The transfer function model in equation (22) may only be an approximation to the dynamic behavior of diesel combustion. However, this simple model is more suitable for control system design than the full physical model (Wellstead and Zanker, 1981). To simulate the process, equation (22) was inverted into the time domain as a differential equation and this equation was solved numerically by the fourth-order Runge-Kutta method (Appendix E).

Simulation procedure

Throughout the simulation study, the open loop test was performed as shown in Figure 9. The flow chart explaining the simulation

procedure is illustrated in Figure 10. To evaluate the identification procedure, four sets of systems with commonly defined parameters were selected. Those system parameters are listed in Table 2. The whole program was written in C and was run on a Zenith personal computer with Turbo C compiler (version 1.5). Simulation starts with reading the process parameters, then the process iterates to yield an output response to the input signal by the process subroutine. After the transient state of the system is passed, the input/output data for steady state is transformed to the frequency domain by the FFT subroutine. The transfer subroutine calculates the transfer function using the transformed input/output signal and the optimization subroutine finally determines the values of process parameters.

Simulated noise

To study the effects of noise on system identification, noise was generated with random numbers, then filtered by a digital low pass filter and scaled properly before use. Program listing of noise generation is described in the Appendix E. The noise was added to the input signal by maintaining a constant ratio between the signal and the average of absolute noise.

Results and discussion

To verify the identification method described earlier in the chapter, two series of tests were performed using the binary multi-frequency signal obtained. The Runge-Kutta 4th order classic solver

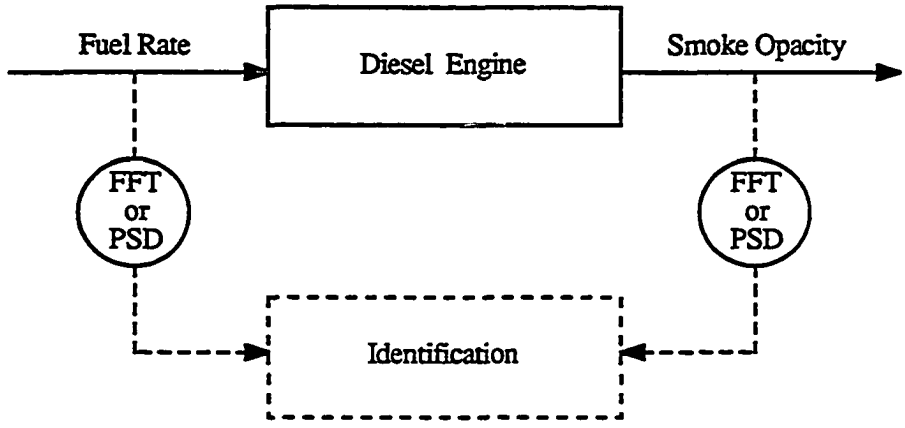


FIGURE 9. Block diagram of the system identification

was implemented with 10 steps per unit sampling time. Both series of tests involved with open loop systems: the first series of tests was applied with noise-free signals and the second series of test included the 10% noise corrupted signals. Those simulation results for both noise-free and noise-corrupted identification are listed in Tables 3 and 4, respectively. Nyquist plots (polar plot) of the open-loop transfer function for each process studied and the frequency response plots of each process are illustrated in Appendix A. In the case of applying noise-free signal, results very comparable to the simulated process parameters were obtained; however, identification in the presence of noise results less comparable to the physical system

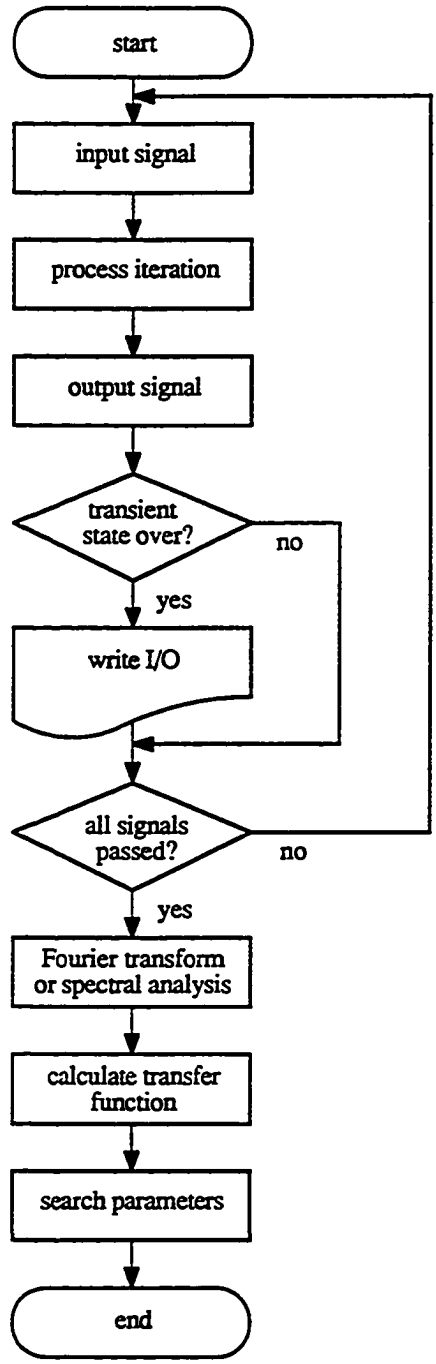


FIGURE 10. The flow chart of simulation procedure

TABLE 2. The parameters of simulated process

System order	Gain	Time constants (min)	Dead time (min)
2nd order	1.0	0.5 1.0	N/A
2nd w/dead	1.0	0.5 1.0	0.3
3rd order	1.0	0.1 0.4 1.0	N/A
4th order	1.0	0.1 0.5 1.0 1.5	N/A

parameter and error was increased as the ratio of noise to signal increased. In terms of control system design, these results were still very informative for finding control parameters, although they were not helpful in determining the actual system model. Sometimes, the noise problem was easily solved by increasing the signal amplitude to the allowable limit in the experiment. Table 3 also shows the identification results of 2nd order with dead time, which is very typical in the diesel combustion process (Wellstead and Zanker, 1981). Generally the effect of dead time on the system has been regarded as difficult to deal with in respect to controller design; however, this study did not show any difference for system analysis between the system with dead time and the one without dead time.

TABLE 3. The results of noise-free identification

System order	Gain	Time constants (min)	Dead time (min)
2nd order	1.0	0.48 1.02	N/A
2nd w/dead	1.0	0.50 1.00	0.3
3rd order	1.0	0.09 0.41 0.99	N/A
4th order	1.0	0.09 0.53 0.94 1.54	N/A

TABLE 4. The results of noisy identification with 10% noise

System order	Gain	Time constants (min)	Dead time (min)
2nd order	1.09	0.48 1.11	N/A
2nd w/dead	1.06	0.60 0.94	0.28
3rd order	0.91	0.00 0.53 0.84	N/A
4th order	1.1	0.09 0.69 0.69 1.81	N/A

ENGINE EXPERIMENT

Diesel Engine and Dynamometer

John Deere four cylinder tractor engine (Model 4239D) studied in this project is naturally aspirated and of the direct injection type. It has 3.92 liters displacement (106 mm bore x 100 mm stroke) and a compression ratio of 17.2. The engine was equipped with a Stanadyne Roosa-Master type DB2 injection pump and pintle type injection nozzles having four 0.28 mm diameter orifices. The start of injection timing based on the injection nozzle lift was factory calibrated at 14±2 degrees before top-dead-center at engine full load and rated speed, while the injectors' nozzle lift pressure was set to 18,600±700 KPa. The manufacturer's specified brake power for the engine at such settings was 56 KW at 2,500 r/min at wide-open-throttle. An A&W Agri-dyno 350, portable, water-cooled dynamometer was used to load the engine. The engine was directly coupled to the dynamometer through a friction clutch. This dynamometer was rebuilt and calibrated by the manufacturer before starting the test. The test engine was cooled by its own radiator. The injection metering pump was modified to directly vary the fuel flow (see following section) and a manual ball valve was placed in the intake manifold to shut off the engine in an emergency situation. Figure 11 shows an overall view of test engine and dynamometer set-up.

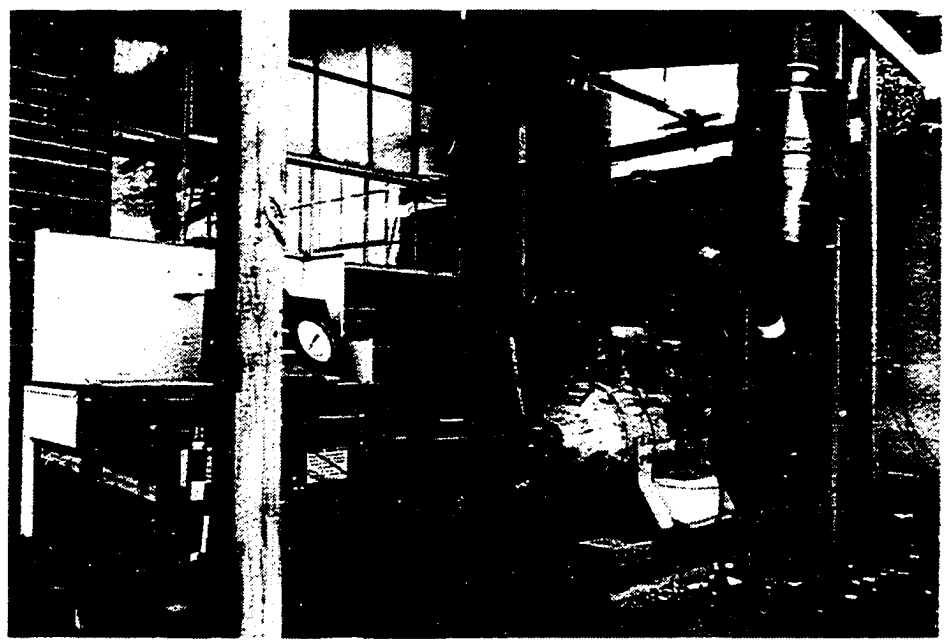


FIGURE 11. Overall view of the test engine and dynamometer set-up.

Data Logger

Hardware

A PC (personal computer) based instrumentation system was used. The computer sends a signal for perturbing the fuel rate, measures and collects the smoke output and temperatures from the engine. The results are stored on floppy disk on a real time basis. In addition to the host computer, the following are included in the system; a multi-purpose interface board to process analog/digital signals, an optical opacity meter for continuous smoke measurement, and a fuel perturbing device. This system also includes instrumentation for temperature, intake air, and fuel rate measurement. Figure 12 shows the block diagram of the data logging system.

Personal computer A Zenith transportable personal computer (Model Z161) based on the Intel 8088 processor was used as a host computer to send input signal, to collect output data, to store results on an on-line basis and to analyze the data to find the diesel engine model for control purpose on an off-line basis. This computer has a 640K byte RAM (Random access, read and write memory), two floppy disk drives, a 8087 math co-processor and a built-in monochrome monitor. The RS-232 compatible serial port mounted on the disk controller-input/output card of this host computer was used for the asynchronous communication with an interface board. Figure 13 shows the details of the wiring between host computer and the Analog Devices μMac-4000. Microsoft MS-DOS Version 3.1 was used as the operating system.

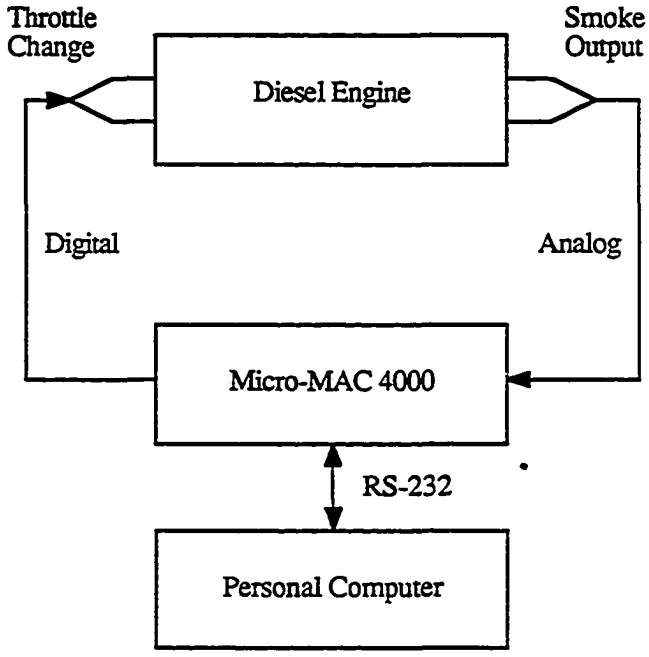


FIGURE 12. Block diagram of data logging system for identification

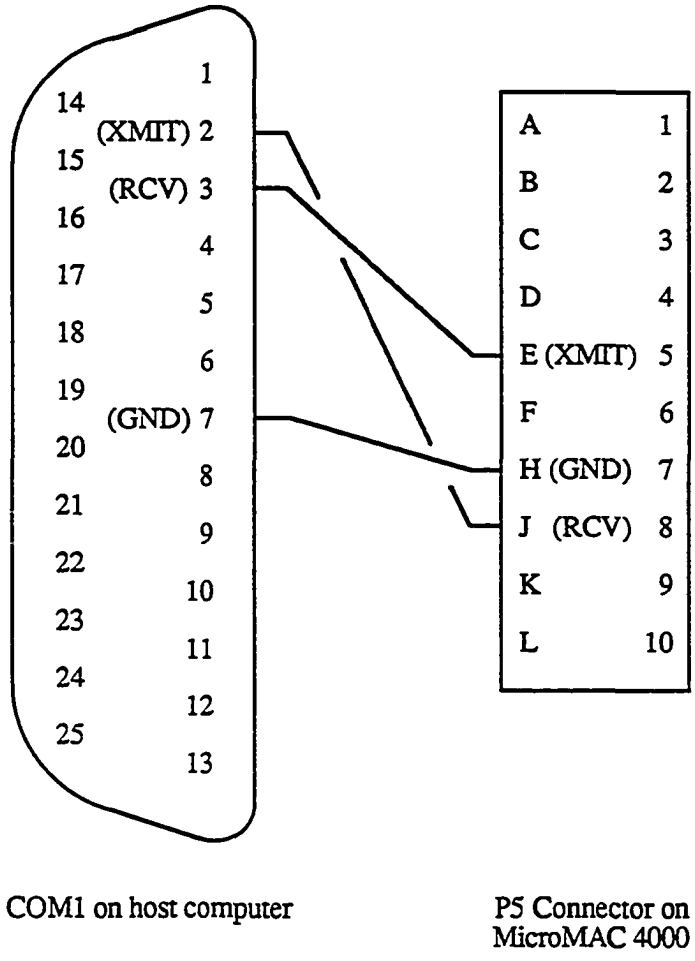


FIGURE 13. Wiring diagram for serial communication

Multi-purpose interface board The μMac-4000, which is a microcomputer based measurement and control system on a single board, provides analog input data process and digital input/output controls; sensor signal conditioning, analog multiplexing, analog to digital conversion, digital I/O and serial communications (Analog Devices, 1981). As the block diagram on the Figure 14 shows, the μMac-4000 has three QMX multiplexer modules. Each module accepts 4 identical analog channels, giving the board a total capacity of 12 analog channels. In addition, the μMac-4000 master board includes the following; an 8-bit 8085A processor based microprocessor with 8K byte of read-only memory(ROM) and 1K byte of random-access memory(RAM), control logic, a universal asynchronous receiver/transmitter (UART), a common analog bus, a programmable gain amplifier, a 13-bit dual slope analog to digital converter, an 8-bit isolated digital input port, an 8-bit digital output port, a power supply and provisions for analog and digital channel expansions. In this study, the board is used to read 5 analog signals for one smoke opacity measurement and four-temperature measurements, and to send the digital signal for fuel flow perturbation. The on-board microcomputer scales, linearizes and converts the input data to engineering units and stores the results in random access memory. The data stored in memory are continuously updated at a minimum rate of 15 or 30 readings per second.

Fuel perturbing device To implement the binary signal as the input variable, the fuel rate must be regulated in a direct manner.

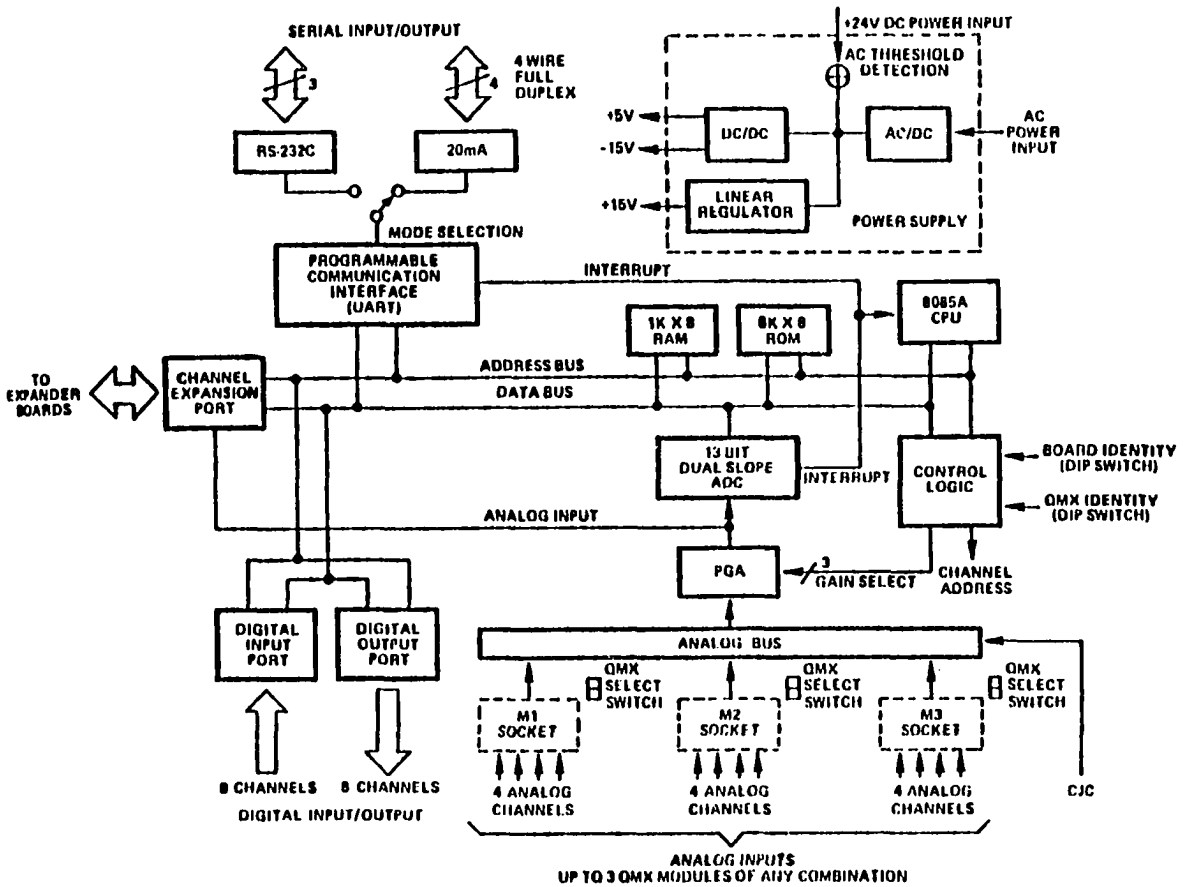


FIGURE 14. Block diagram of μMac-4000 interface board (Analog Devices, 1981)

The current design of injection metering pump, however, adopts the mechanical governor control system in which fuel is regulated by the combination of engine speed and throttle movement. Thus to directly vary the fuel flow rate regardless of engine speed, both governor link mechanism and the spring connected to the metering valve from governor weights in the original injection pump were removed. Then a two-bar linkage was installed between the metering valve and the throttle lever (Figure 15). Thus it became possible to directly regulate the metering valve by the throttle lever. To generate fuel rate perturbation, two solenoids with a power amplifying board were used to reciprocate the extended throttle lever. These solenoids were controlled by signals generated by the host computer. Two solenoids (Guardian Elec.) were mounted on a plate whose location could be linearly changed to the desired operating point of diesel engine (speed and torque) (Figure 16). This fuel perturbing device was calibrated by the weight method to find the relation between the throttle lever position vs. the actual fuel flow rate.

Smoke measurement A portable opacity smoke meter (Telonic Berkeley Model 200) was used to measure the opacity of the exhaust smoke continuously. This system includes a sensor assembly and an indicator unit shown on the functional block diagram in Figure 17. The light source is a pulsed, solid-state green light-emitting diode which transmits light to a silicon photo detector. The meter has 0.01s response time to reach the 90% of the full scale and less than 2.5%

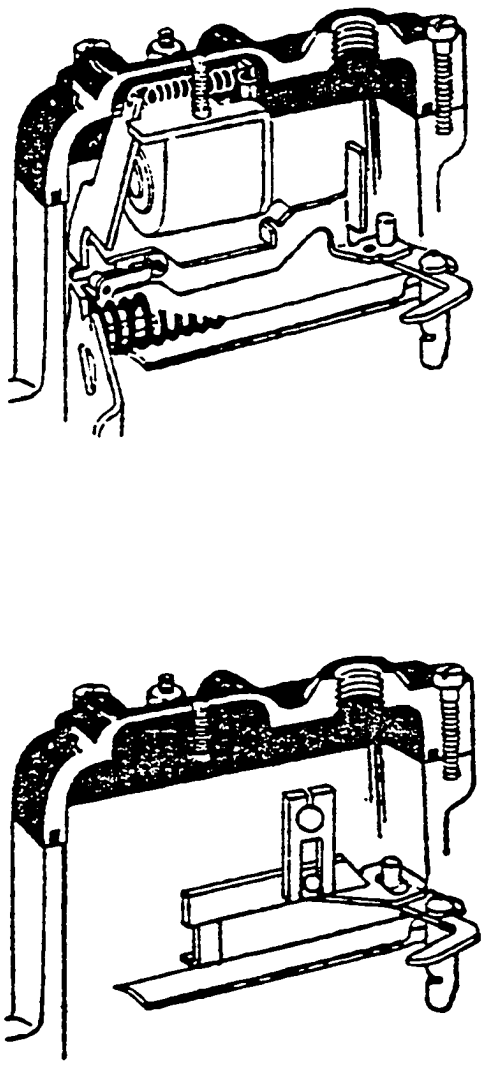


FIGURE 15. Section views of injection metering pump: Before modification (upper) and after modification (lower)

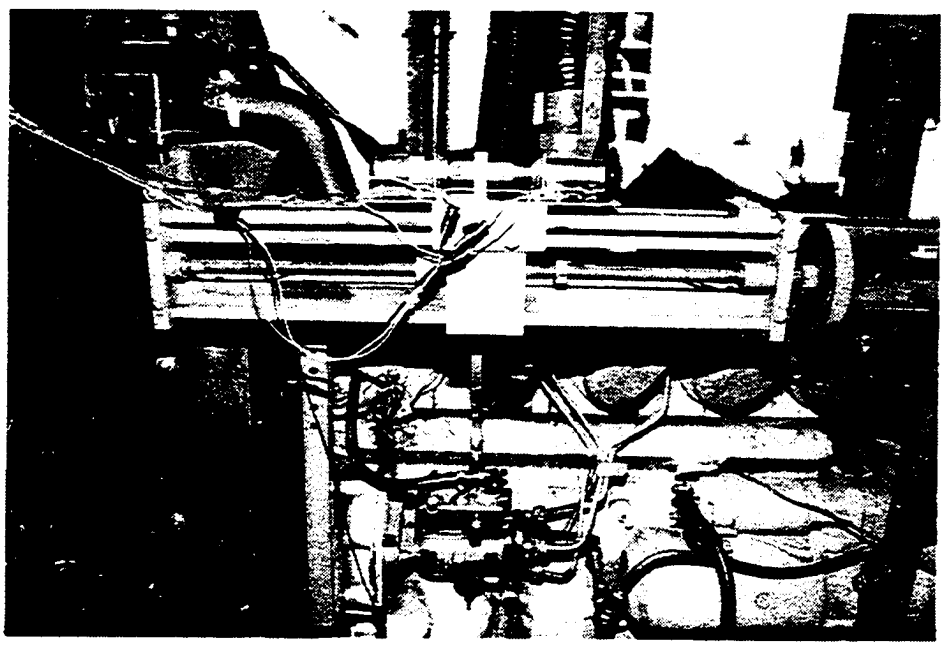


FIGURE 16. Overall view of fuel perturbing system

drift per hour. A gas temperature correction feature compensates the smoke density reading, $K(m^{-1})$ to standard 100 °C gas temperature. Exhaust smoke level was expressed as opacity, % or density, $K(m^{-1})$ on its own indicator and as opacity, fraction of volt on the computer screen, which is connected separately through μMac-4000 interface board. The percent opacity was used as a unit of smoke throughout this study.

Temperature measurements Temperatures at various points on engine were measured using locally fabricated ANSI type K (Chromel-Alumel) thermocouples. These were placed to measure the temperature at the following locations in the engine:

1. Lubricating oil temperature at the engine oil sump.
2. Coolant temperature at the inlet housing of the coolant pump.
3. Intake air temperature at the intake manifold.
4. Exhaust gas temperature at the exhaust manifold.

These four thermocouples were then connected to the QMX03 module in the μMac-4000 interface. Temperature readings were presented on the monitor of the host computer from the control program and readings were recorded on floppy disk when smoke data were collected. Two sets of calibration were made: an accurate voltage input was applied to the analog channel of the interface board as recommended in the user's manual (Analog Devices, 1981) and various known temperatures (0 - 90 °C) were applied to the thermocouples.

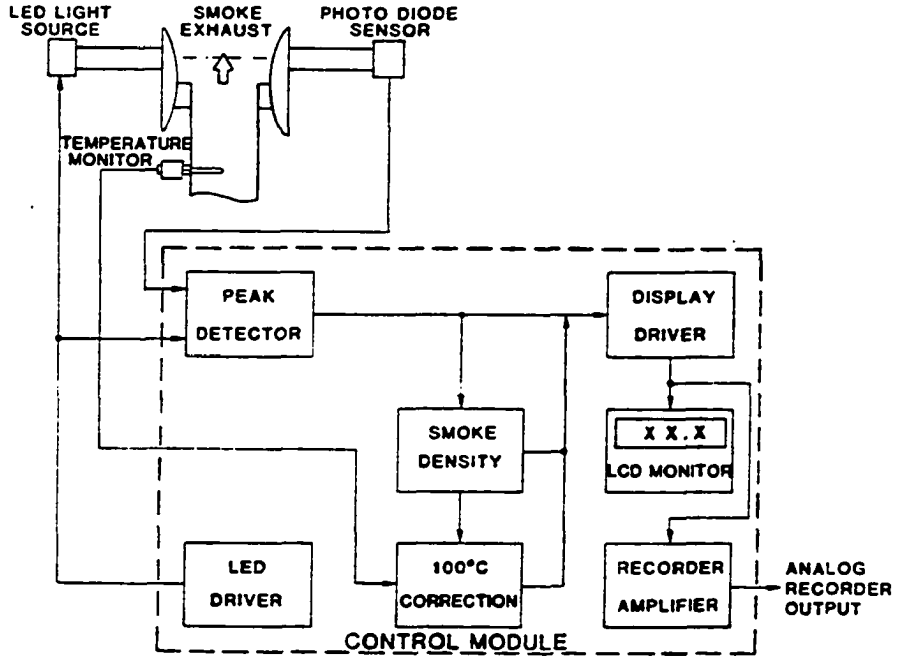


FIGURE 17. Functional block diagram of model 200 smoke meter (Berkely Controls, 1988)

Fuel flow measurement Fuel flow measurement was performed by weighing the fuel reservoir over a given period of engine operation and by dividing the weight difference of fuel used by the elapsed time. An electronic balance having a maximum reading of 20 kg and resolution of 5g and a digital stop watch were used.

Air flow measurement A laminar flow element (Meriam model 50 MC2-4) with a manometer was used to measure the engine intake air flow rate.

Software

The instrumentation system software for identification was written in C language with one internal function and five external functions written in assembly language for the real-time operation and compiled on Turbo-C compiler, version 1.5, and Microsoft macro assembler, version 4.0, respectively. The software is designed to control the μMac-4000 interface board in order to send the digital signal for throttle oscillation, to collect and sort the output data to floating-point values and, finally, to store those on the floppy disk to make them available for later analysis. All communication was performed on a real time basis. The simplified expression is shown on the flow diagram of the Figure 18, where the command with an asterisk is written by assembly language. The details of each subroutine are described in the following subsections and program listings are shown on Appendix F.

Real time operation and sampling interval An assembly routine was written for the real time operation by programming of timer chip

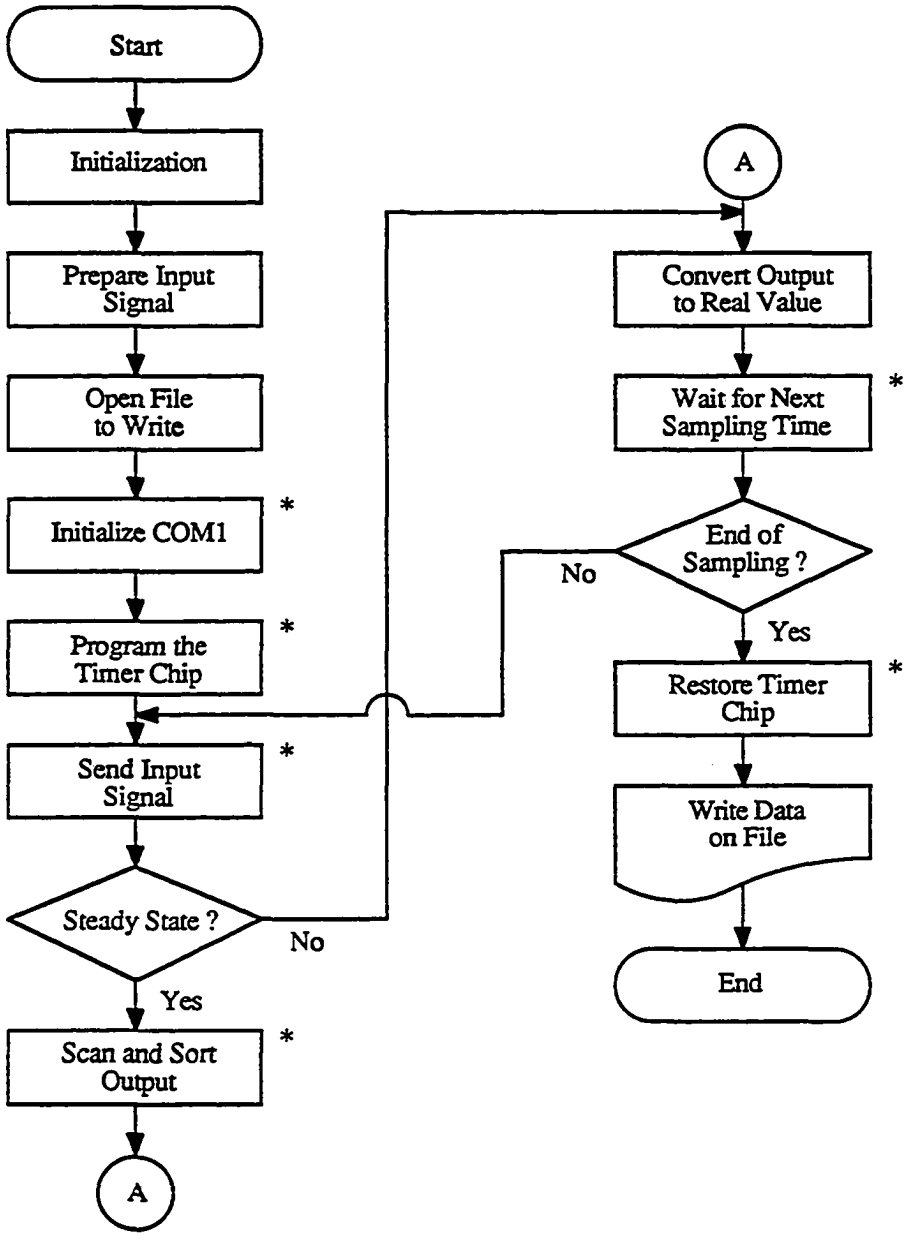


FIGURE 18. Flow chart of data logging routine

(8253) on the host computer. Since the input clock rate to timer chip is 1.19318 million times per second, and the largest number held by 16-bit register is 65535, the output clock pulse rate from the timer chip (8253) becomes 18.20648193 times per second. With this real-number clock pulse rate, the precise control of sampling time is difficult to achieve, for example, 100 ms sampling interval requires 1.8206 times pulse rate which is practically impossible to count. Thus the 8253 timer chip on the host computer is programmed to replace the original latch value of 65535 with the faster value of 11931 which results in the change of the clock pulse rate from 18.206 to 99.998 times per second and 10-pulses duration, for example, yields 100 ms sampling interval. With this new latch value, the period between clock pulses becomes 10 ms.

Since this system can not collect samples faster than 62 ms due to hardware limits, mostly from communication with interface board, the sampling rate was set to 70 ms for binary signals and to 200 ms for step tests.

Communication with μMac-4000 Five assembly modules and four subroutines were written for serial communication with the multi-purpose interface board, μMac-4000. Those include initialization of the serial port of host computer (COM1) with parameters shown on Table 5, for the communication with μMac-4000, transmission of μMac-4000 commands for control, reception of the response of the interface board to these commands, and removal of unwanted charters from the μMac-4000

reply. Table 6 shows the μMac-400 commands with the response used in this study and Figure 19 shows the flow chart of the sorting algorithm used for the removal of unwanted characters from the μMac-4000 reply.

TABLE 5. Parameters used in serial communication

Parameters	Value
Word length	7 bits
Stop bits	2 bits
Parity	Even
Baud rate	9600 bps

TABLE 6. μMac-4000 command and response

Purpose	Command	Response
To send input signal thru. bit 7 on port 1	*0:SET/1/7:(cr)[a] *0:CLE/1/7:(cr)[a]	*00::(cs)[b](cr)[a](lf)[c] *00::(cs)[b](cr)[a](lf)[c]
To receive smoke data from channel 0	*0:CHA/0:(cr)[a]	*00:(data)[d]:(cs)[b] (cr)[a](lf)[c]
To receive temperature from channel 8 - 11	*0:SCA/8/11:(cr)[a]	*00:(data)[d]/(data)/ (data)/(data):(cs)[b] (cr)[a](lf)[c]

[a]Carriage return.

[b]Sum of ASCII code of each character received.

[c]Line feed.

[d]7 bytes data including a decimal point and a sign.

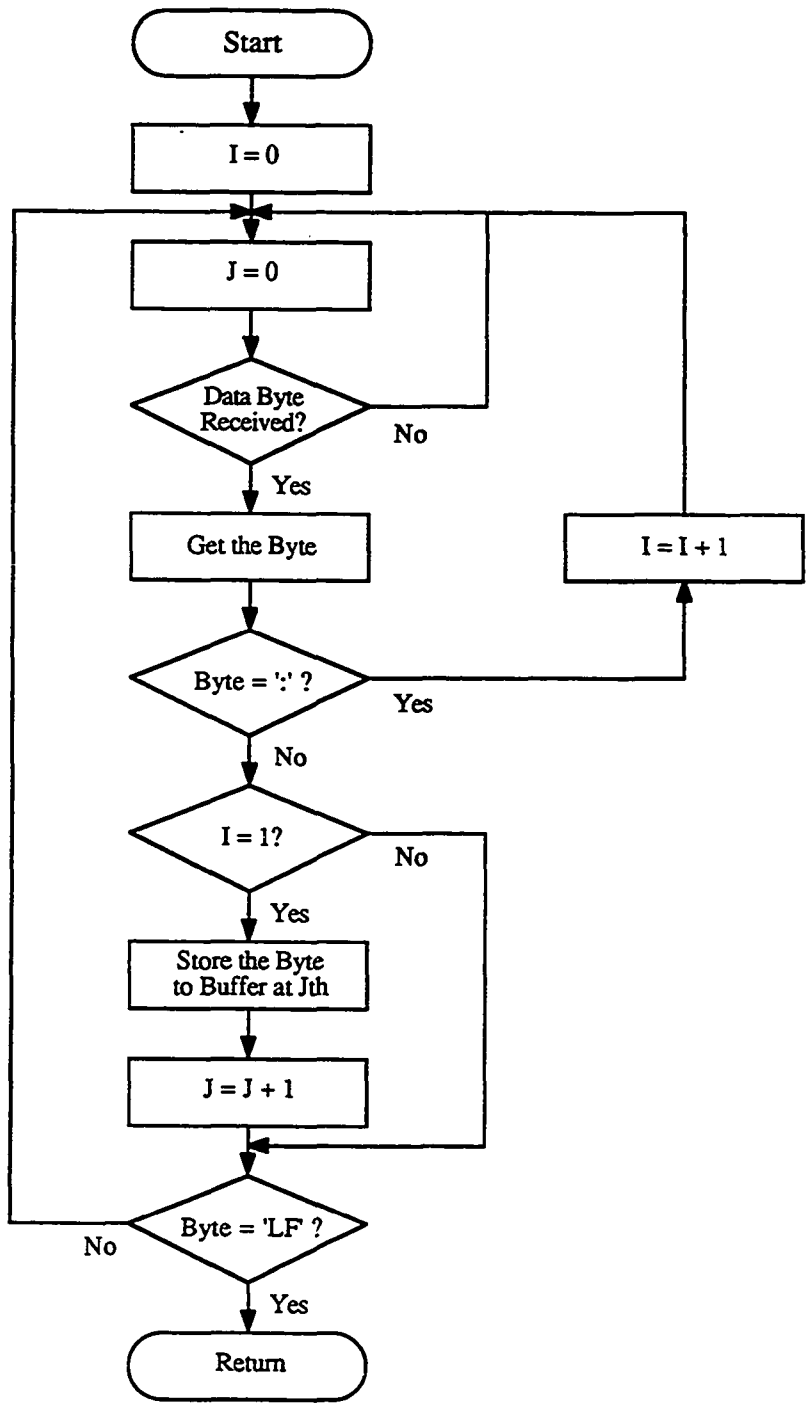


FIGURE 19. Flow chart of output data sorting routine

Model building package

To perform the model building task in a simple and convenient way, a menu driven software package was developed by adding the set up program and the graphic generating program to the data collection routine explained in the previous section. This package consists of 3 main tasks which have their branch tasks as shown in Figure 20. The help session for brief explanation of general usage is included as follows:

Set up This menu is made for the preparation of the main procedure and of three subtasks; one is for test identification for the appropriate file generation and the next is for testing the major equipment, and the last is for temperature display. In the identification menu, the filename for each set of tests will be assigned by selecting a desired speed, a torque, a replication number, and an input signal. Verification is allowed to check if a proper menu was selected. In the pre-test menu, the fuel perturbation can be varied by changing the input signal manually using the arrow key on the keyboard and smoke output can be displayed on the monitor. In the 'check temperature' menu, four different temperature readings will be displayed on the monitor continuously.

Data collection The data collection with signal transmission for modelling will be performed in this task and the collected data will be written on the floppy disk. The status of data collecting

process will be displayed on the monitor. The details of this program are explained in previous section and prgram listings are included in Appendix F.

Graphics To see the collected data in convenient way, time domain graph was displayed by selecting this menu (Appendix F).

Data Collection

Data collection was performed at 9 different engine operating points (Table 7) which were intended to be representative of a typical engine application. At each operating point, three different input signals, including step, pseudo-random binary and multi-frequency binary signals, were applied for 9 consecutive periods and the average of 8 periods, from 2nd to 9th period, was recorded on the diskette.

Results and Discussion

The smoke opacity response on the fuel metering rate in time domain is shown in Appendix B. Those Figures in Appendix B include input signal shape for reference and actual fuel rates are shown in Table 8 through 10. These Tables also show the actual speed variation and air-fuel ratio at each operating point.

Step response

At the engine speeds of 1000 r/min and 1500 r/min, the smoke opacity increased very rapidly right after fuel rate was increased and fell back quickly, and the smoke level decreased slowly to the steady

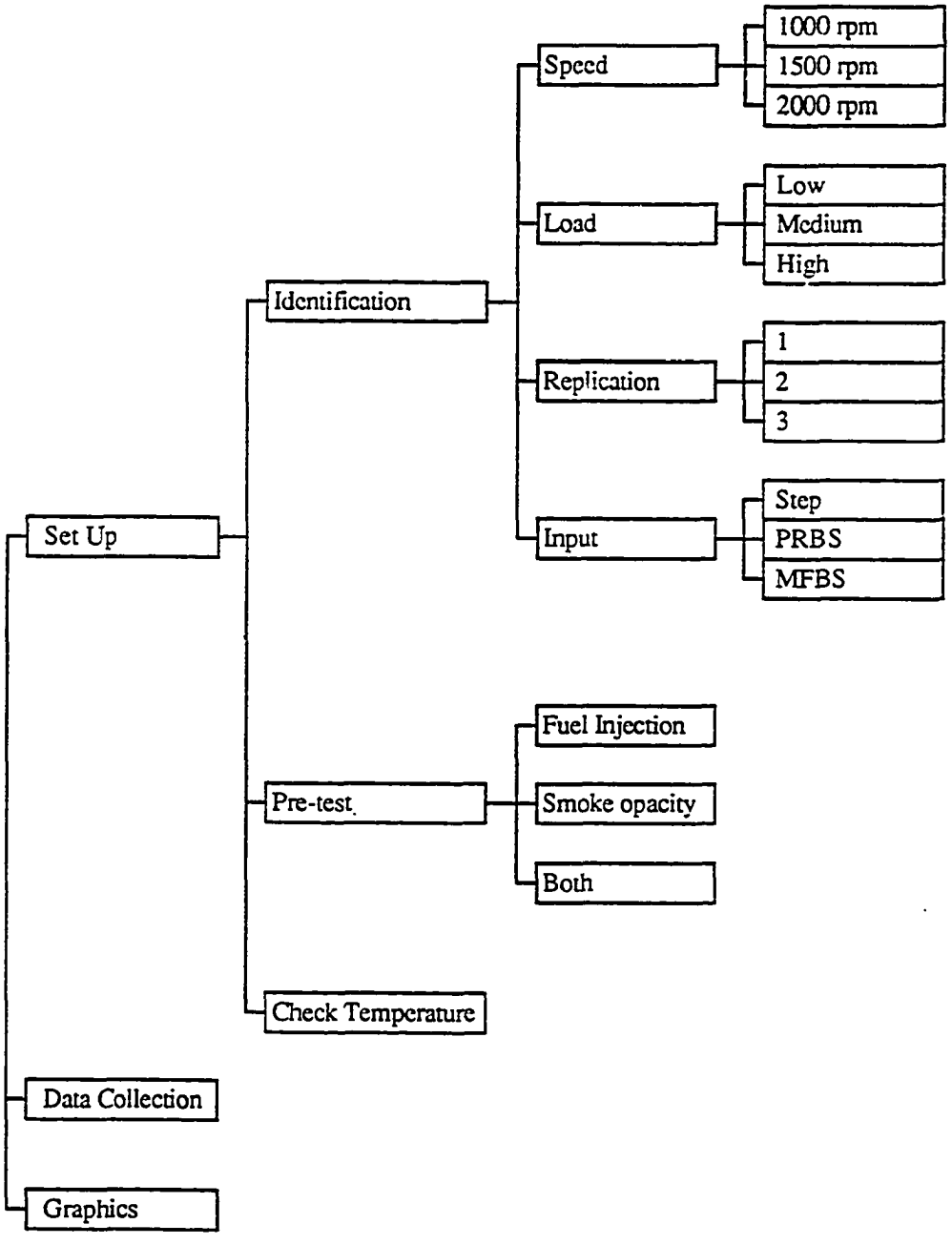


FIGURE 20. Menu command structure of a model building package

TABLE 7. Engine operating matrix

Speed (r/min)	Load (N-m) Low	Medium	High
1000	81.40	122.10	162.80
1500	81.40	122.10	162.80
2000	81.40	122.10	162.80

TABLE 8. Actual engine speed and air fuel ratio at 1000 r/min

Load (N-m)	Speed (r/min)	Air flow (g/min)	Fuel flow (g/min)	Air-Fuel ratio
81.40	1120	2292	55.0	41.67
(low)	800	1608	39.0	44.67
122.10	1150	2250	74.0	30.41
(medium)	980	1790	53.0	33.77
162.80	1150	2285	88.3	25.88
(high)	920	1876	66.7	28.13

TABLE 9. Actual engine speed and air fuel ratio at 1500 r/min

Load (N-m)	Speed (r/min)	Air flow (g/min)	Fuel flow (g/min)	Air-Fuel ratio
81.40	1640	3056	87.5	34.93
(low)	1330	2859	68.8	41.59
122.10	1670	3507	107.5	32.62
(medium)	1430	3029	86.3	35.15
162.80	1980	4144	146.7	28.25
(high)	1310	2809	85.0	33.05

TABLE 10. Actual engine speed and air fuel ratio at 2000 r/min

Load (N-m)	Speed (r/min)	Air flow (g/min)	Fuel flow (g/min)	Air-Fuel ratio
81.40	2430	5022	104.0	48.29
(low)	1910	4090	71.3	57.36
122.10	2395	4916	132.9	36.99
(medium)	1830	3863	96.5	40.03
162.80	2360	4817	161.7	29.79
(high)	1750	3678	121.7	30.22

TABLE 11. Engine temperature at various locations

Speed (r/min)	Load (N-m)	Engine oil (°C)	Coolant (°C)	Intake air (°C)	Exh. gas (°C)
1000	81.40	78.6	57.1	21.6	212.0
	122.10	82.3	53.9	22.6	254.1
	162.80	85.1	50.5	23.2	319.8
1500	81.40	76.0	55.0	10.9	233.5
	122.10	82.7	53.9	12.2	288.5
	162.80	86.6	52.6	15.1	319.1
2000	81.40	83.1	48.7	12.2	270.4
	122.10	89.5	53.1	13.8	314.5
	162.80	94.2	53.2	16.2	355.5

state level. During the early period, the sharp increase of smoke level is due to the decrease of air-fuel ratio and at the next period, the effect of the increased injection pressure cancels out the effect of air-fuel ratio. As the fuel rate decreased, same trend was observed. It is interesting to note that the steady state level of the smoke at higher fuel rate is lower than that at the lower fuel rate. This is due to the effect of injection pressure. Thus the effect of air-fuel ratio on smoke has only a transient effect, however, the lower fuel injection pressure is responsible for higher smoke level in the speed range.

At 2000 r/min, the smoke opacity was initially increased very rapidly and continued to increase rather slowly to the steady state level. The steady state level of smoke increased with increasing fuel

rate due to the decreased air-fuel ratio. The change of injection fuel pressure did not affect the smoke production because the injection pressure at this speed was expected to be much higher than the critical injection pressure, which was responsible for higher smoke due to locally very low air-fuel ratio.

At engine speeds of 1000 r/min and 1500 r/min, it took approximately 50 s to reach the steady state smoke level and this time was similar at different load levels. At 2000 r/min, the time taken became much shorter and varied with the change of load.

Frequency response

To obtain the frequency response of smoke opacity on fuel injection rate when perturbed by binary signal, power spectral density method and fast Fourier transform technique (Appendix H) described in the previous chapter were applied. These techniques were applied to the pseudo-random binary data and to the multi-frequency binary data. As shown in Figure 21, both amplitude and phase response on pseudo-random binary signal have much scattered data points at the high frequency range. The power spectral density method was tested with the pseudo-random binary signal and corresponding output data for an assumed process, and similar trend was observed as shown in Figure 22. The reason for this scatter produced by the power spectral density method is possibly due to sampling time, however, it is obvious that the scattered points at high frequency on the frequency response from experiments were not only due to the measurement noise but to the

numerical effects. Thus only those data in the lower side frequency range, from 1st to 36th data pair, were used for curve fitting purposes.

Frequency response results obtained using the multi-frequency binary signal did not provide information good enough to derive transfer function from them. This was because of severe noise due to cycle and cylinder variations, and the possibly due to the slow sampling interval. For these reasons, the frequency response using the pseudo-random binary signal was used to find the transfer function between smoke opacity and fuel metering rate.

Through observation of the phase response of each data set, four different sets of postulated transfer functions which have the form of equation (23) were selected.

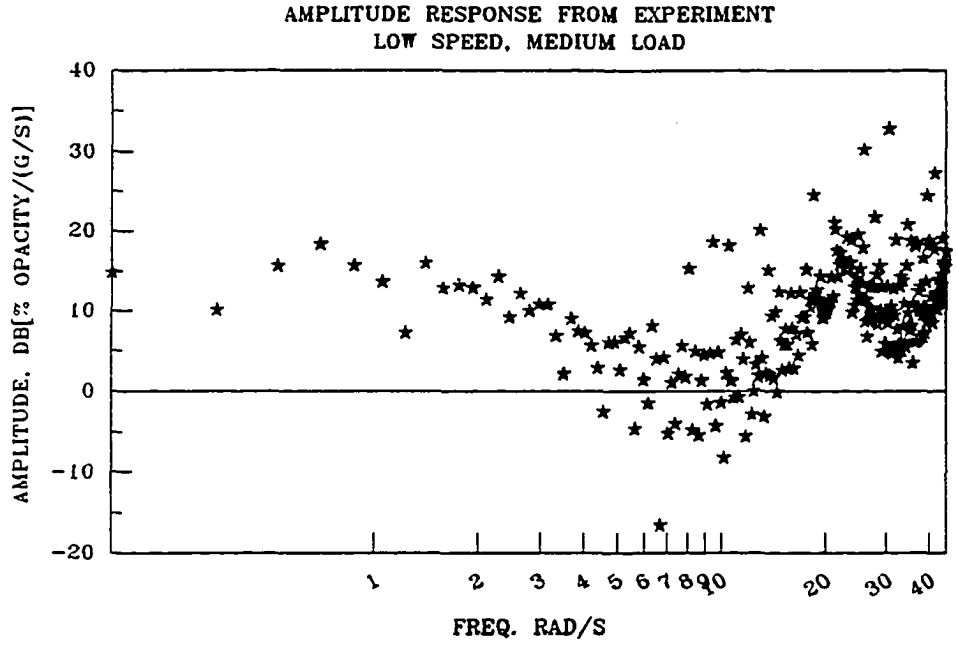
$$T(s) = \frac{G(aS+1)\ e^{-Ds}}{(b_1S+1)(b_2S+1).....(b_n+1)} \qquad (23)$$

where a, b_n : Time constants

D : Dead time

T(s) : Transfer function

Box's direct search method described in the previous chapter was then used to find the parameters of the transfer function. The search program was run more than ten times for each postulated transfer



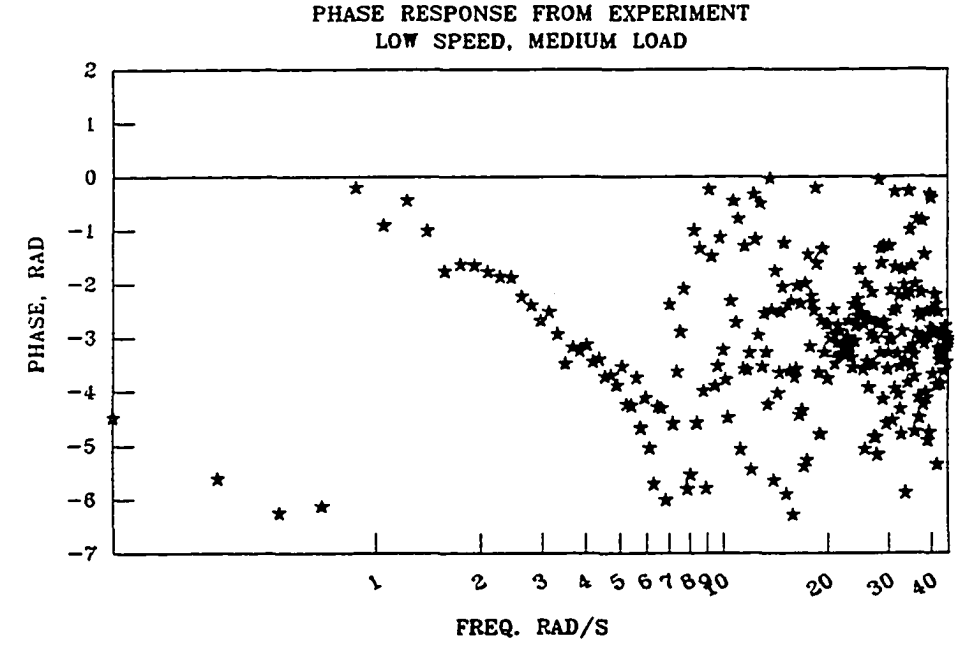


FIGURE 21. Typical frequency response obtained from experimental data by the power spectral density method

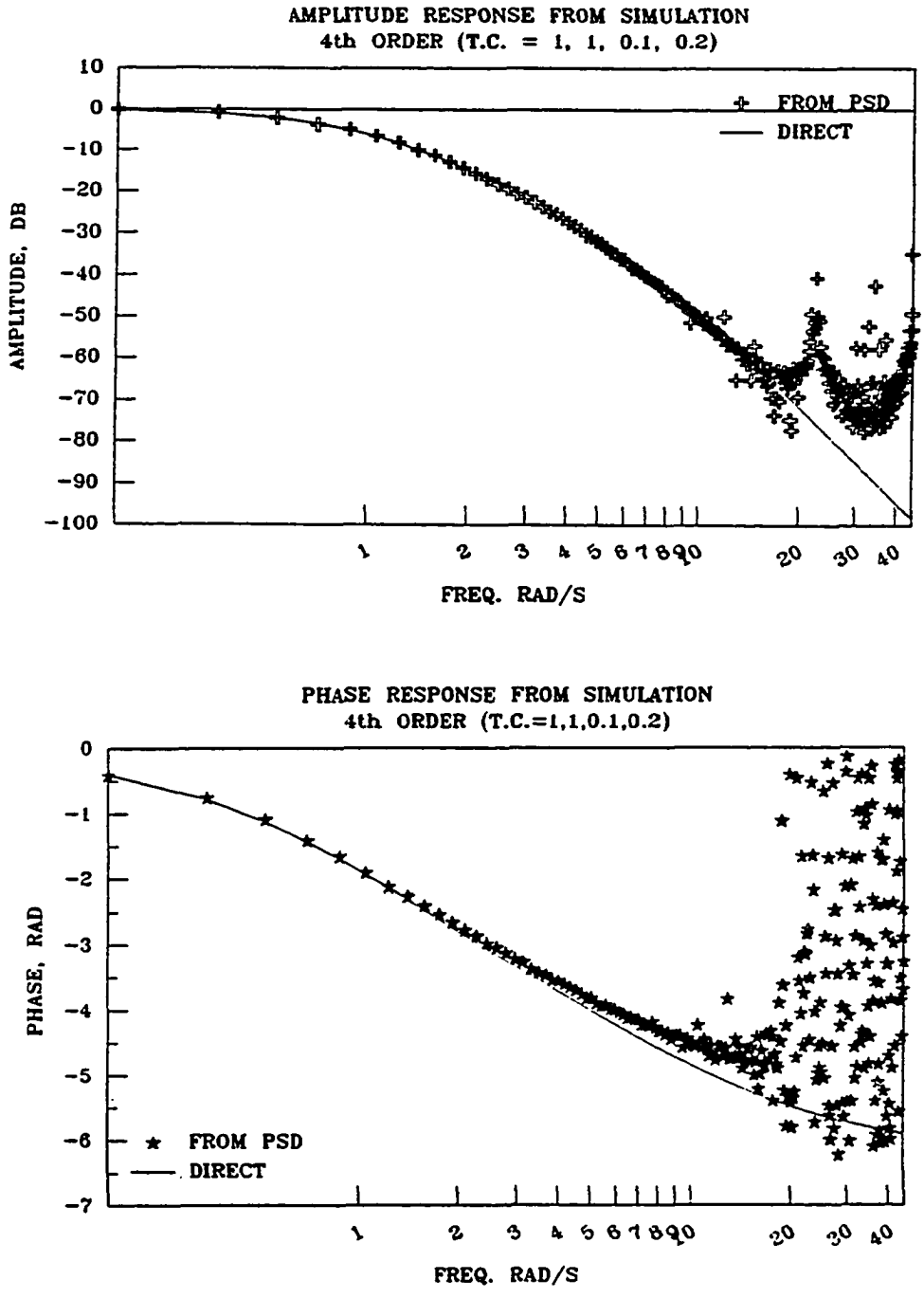


FIGURE 22. Sample frequency response obtained from assumed data by the power spectral density method

function and the best set of parameters was chosen. The final sets of parameters are shown in Table 12 through Table 14. The frequency responses from the experimental data and the best fit transfer functions for 9 different operating points are shown in the Appendix C.

TABLE 12. Parameters and object function value of transfer function at 1000 r/min

Load	Order	Gain[a]	Dead time (s)	Num. (s)	Time constants Denominator (s)				Value of object function (Eq. 21)
Low	3	1.2975	0.0000	0.9998	0.4106	0.4094	0.4066		395.17
	3 w/d	1.041	0.4617	0.9994	0.2592	0.2592	0.2564		365.29
	4	1.4235	0.0000	0.9999	0.3049	0.3064	0.3076	0.3058	378.19
	4 w/d	1.071	0.3501	0.9997	0.2413	0.2087	0.2052	0.2143	363.22
	4[b]	5.9205	0.0000	0.0268	0.4679	0.4634	0.4666	0.4666	241.63
Med.	3	4.7272	0.0000	0.0	0.2841	0.2888	0.2878		234.07
	3 w/d	3.3678	0.4261	1.0000	0.3687	0.3653	0.3649		151.66
	4	4.2786	0.0000	1.0000	0.3907	0.3922	0.3911	0.3929	193.65
	4 w/d	3.3873	0.2889	1.0000	0.3090	0.2961	0.2976	0.2970	148.10
High	3	4.9328	0.0000	0.0	0.2965	0.2961	0.2974		491.94
	3 w/d	3.4080	0.4769	0.9998	0.3463	0.3475	0.3531		373.86
	4	4.4757	0.0000	0.9999	0.3990	0.3976	0.3968	0.4006	446.51
	4 w/d	3.440767	0.3432	1.0000	0.2855	0.2856	0.2842	0.2865	370.90

[a]Unit = [% opacity/fuel rate(g/s)].

[b]Has a additional numerator S term.

TABLE 13. Parameters and object function value of transfer function at 1500 r/min

Load	Order	Gain[a]	Dead time (s)	Num. (s)	Time constants Denominator (s)			Value of object function (Eq. 21)
Low	2	0.3029	0.0000	0.0001	0.3215	0.3182		8.77
	2 w/d	0.1662	0.4292	1.3087	0.4701	0.4783		8.49
	3	0.1964	0.0000	1.9012	0.5296	0.5053	0.5050	8.63
	3 w/d	0.1341	0.2938	1.7895	0.2007	0.4736	0.3819	8.44
Med.	2	0.3243	0.0000	0.0001	0.3592	0.3595		3.57
	2 w/d	0.2066	0.4888	0.9999	0.4508	0.4542		3.07
	3	0.3215	0.0000	0.0001	0.2438	0.2257	0.2284	3.36
	3 w/d	0.2140	0.3391	0.9977	0.3026	0.3695	0.2887	3.01
High	2	0.2089	0.0000	0.0	0.3481	0.3487		1.14
	2 w/d	0.1385	0.5169	0.9999	0.4119	0.4224		0.80
	3	0.1935	0.0000	0.9998	0.4719	0.4807	0.4728	1.04
	3 w/d	0.1439	0.3717	0.9996	0.3199	0.2882	0.3158	0.77

[a]Unit = [% opacity/fuel rate(g/s)].

TABLE 14. Parameters and object function value of transfer function at 2000 r/min

Load	Order	Gain[a]	Dead time (s)	Num. (s)	Time constants Denominator (s)				Value of object function (Eq. 21)
Low	3	0.3343	0.0000	2.882	0.6546	0.6145	0.6087		3.69
	3 w/d	0.2991	0.2782	1.9873	0.4495	0.2522	0.5827		2.95
	4	0.473430	0.0000	0.9989	0.3322	0.3355	0.3399	0.3278	3.28
	4 w/d	0.4694	0.2479	0.3999	0.1943	0.1934	0.1963	0.1962	3.17
Med.	3	1.8527	0.0000	0.0	0.2744	0.2744	0.2744		2.37
	3 w/d	1.6599	0.1736	0.0000	0.1629	0.1629	0.2994		1.11
	4	1.7351	0.0000	0.0	0.1965	0.1965	0.1965	0.1965	1.36
	4 w/d	1.6645	0.1161	0.0000	0.1207	0.1206	0.1207	0.3215	1.10
High	3	2.0820	0.0000	0.0	1.0000	0.3365	0.3172		8.78
	3 w/d	1.9791	0.2943	0.0000	1.0000	0.3336	0.0001		8.24
	4	2.0064	0.0000	0.0	0.2018	0.2031	0.2026	1.0000	8.54
	4 w/d	1.9755	0.2916	0.0000	0.0002	0.3343	0.0022	0.9998	8.24

[a]Unit = [% opacity/fuel rate(g/s)].

SUMMARY AND CONCLUSION

The objective of this research was to develop an empirical model of a diesel engine transfer function for control purposes by the system identification method. With the assumption of linear diesel operation in a limited region, the linear theory was adopted for system identification. Step input signals and binary input signals (pseudo-random binary sequence and multi-frequency binary sequence) were selected as input signals. The system identification method using binary signals was tested by computer simulation before applying it to the engine experiment. To perform this research, an engine control and monitoring package was developed using an 8088 based personal computer. This package included a host computer, a multipurpose interface board for signal processing, a fuel metering valve perturbing mechanism with a modified fuel injection pump, and an operating software written in C and Assembly languages. Temperatures at various engine locations and air-fuel flow were also measured in this study.

A step response study showed that the effect of air-fuel ratio was more important than the injection pressure in transient state smoke. This was also true in steady state when the injection pressure was higher than the critical value.

Data analysis was performed by frequency response method using fast Fourier transform for multi-frequency binary data and by spetral analysis for pseudo-random binary data. The resulting frequency domain data were fitted using a nonlinear optimization procedure to find the model parameters.

The higher order of transfer function between smoke opacity and fuel metering rate obtained for each operating point in this study can be utilized in development of an electronic injection pump for the reducing smoke level and increasing fuel economy. With fast computation, on-line identification might be possible and adaptive control could be implemented. Adaptive control with on-line identification and appropriate sensors leads to an integrated engine management system.

In summary:

1. The system identification using binary signals, such as pseudo-random and multi-frequency binary signal was implemented to find the transfer function between fuel metering rate and smoke opacity.
2. A personal computer based data logger for system identification purpose was developed to find the empirical model for a particular diesel engine.
3. System identification was verified and showed a reasonable match between simulation and experiment.
4. Nine sets of transfer functions were obtained.
5. In this experiment, the pseudo-random binary input signal yielded better results than the multi-frequency binary input signal.

RECOMMENDATION

To improve the quality of the model for the better control of smoke production, several recommendations can be given:

1. Since smoke production depends on many operating and design factors and also relates to other gaseous emissions such as hydrocarbon and nitric oxide, multiple input/multiple output model may be more suitable.
2. An increased sampling frequency will provide flexibility in data handling so that model quality can be improved. This can be accomplished by adopting a faster signal actuator, a signal processor and a sensor with a response time of less than 20 ms.
3. With multi-tasking software and more than two computers, input signal transmitting, data sampling and computation can be performed simultaneously

BIBLIOGRAPHY

Akaike, H. 1981. Modern development of statistical methods. In P. Eykhoff, Ed. Trends and progress in system identification. Pergamon Press, New York.

Alkidas, A. C. 1984. Relationships between smoke measurements and particulate measurements. SAE Paper No. 840412.

Amann, C. A., D. L. Stivender, S. L. Plee, and J. S. MacDonald. 1980. Some rudiments of diesel particulate emissions. SAE Paper No. 800251.

Analog Devices. 1981. User's manual-μMac-4000. Analog devices Inc., System components division, Norwood, Ma.

Astrom, K. J., and P. Eykhoff. 1971. System identification-A survey. Automatica, 7: 123-162.

Backhouse, R., and D. E. Winterbone. 1986. Dynamic behavior of a turbocharged diesel engine. Paper No. 860453. SAE Trans., 95, sec. 3: 73-80.

Balakrishnan, A. V., and V. Peterka. 1969. Identification in automatic control systems. Automatica, 5: 817-829.

Berkely Controls. 1988. Operation and service manual-Model 200 smoke meter. Berkeley Controls, Inc., Laguna Beach, California.

Black, O. P. and W. E. Scahill. 1983. Audels diesel engine manual. Third Ed. The Bobbs-Merrill Co., Inc. New York.

Bowns, D. E. 1970. The dynamic transfer characteristics of reciprocating engines. Proc. Instn. Mech. Engrs. 1970-1971, 185, 16/71: 185-201.

Box, M. J. 1965. A new method of constrained optimization and comparison with other methods. Comput. J., 8, No. 1: 42-52.

Briggs, P. A. N., K. R. Godfrey, and P. H. Hammond. 1967. Estimation of process dynamics characteristics by correlation methods using pseudorandom signals. IFAC symp. on Identification and Process Parameter Estimation, Prague, Czechoslovakia.

Bryzik, W. and C. O. Smith. 1978. Relationships between exhaust smoke emissions and operating variables in diesel engines. SAE Paper No. 770718.

Cadzow, James A. and Hugh F. Van Landingham. 1985. Signals, systems and transforms. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Cassidy, J. F. 1978. A state variable model for engine control studies. GMR report ET-180.

Chang, M. -F. and J. A. Sell. 1982. A linearized model of engine torque and carbon monoxide emissions. SAE Paper No. 830927.

Chen, C. F., and I. J. Hass. 1968. Elements of control system analysis. Prentice Hall, Englewood Cliffs, New Jersey.

Chirlian, P. M. 1969. Basic network theory. McGraw-Hill, New York, NY.

Cooley, J. W. and J. W. Tukey. 1965. An algorithm for machine calculations of complex Fourier series. Math. Comput., 19, No. 90: 297-301.

Cramer, H. 1946. Mathematical methods of statistics. Princeton Univ. Press, Princeton, NJ.

Cumming, I. G. 1970. Frequency of input signal in identification. Proc. of the 2nd IFAC Symp. on Identification and Process Parameter Estimation. Paper 7.8, Prague, June.

Cusset, B. F. and D. A. Mellichamp. 1975. On-line identification of process dynamics. A multi-frequency response method. Ind. Eng. Chem. Process Des. 14, No. 4: 359-368.

Davies, W. D. T. 1970. System identification for self-adaptive control. Wiley-Interscience, London.

Dent, J. C., and P. S. Mehta. 1981. Phenomenological combustion model for a quiescent chamber diesel engine. SAE Paper No. 811235.

Dolan, D. F., and D. B. Kittleson. 1978. Diesel exhaust aerosol particle size distributions-Comparison of theory and experiment. Paper No. 780110. SAE Trans., 87: 462-468.

Eykhoff, P. 1974. System identification: Parameter and state estimator. Wiley-Interscience, New York.

Eykhoff, P. 1981. Trends and progress in system identification. Pergamon Press, New York.

Fasol, K. H. and H. P. Jorgl. 1980. Principles of model building and identification. Automatica, 16: 505-518.

Fruchete, R. D. and A. Kate. 1978. Transfer function modelling of a gasoline engine and engine actuator. GMR memorandom 53-46. General Motors Research Laboratories, Warren, MI.

Godfrey, K. R. 1970. The application of pseudorandom sequences to industrial process and nuclear power plant. 2nd IFAC Symp. on Identification and Process Parameter Estimation, Paper 7.1, Prague, Czechoslovakia.

Godfrey, K. R. 1980. Correlation methods. Automatica, 16: 527-534.

Goodwin, G. C. 1987. Experiment design for system identification. In M. Singh Ed., Encyclopedia of systems and control. Pergamon Press, Oxford.

Goodwin, G. C., and R. L. Payne. 1977. Dynamic system identification. In Experimental design and data analysis. Academic Press, New York.

Greeves, G. 1979. Response of diesel combustion systems to increase of fuel injection rate. SAE Paper 790037.

Groblicki, P. J., and C. R. Begeman. 1979. Particle size variation in diesel car exhaust. SAE Paper No. 790421.

Gustavsson, I. 1975. Survey of application of identification in chemical and physical process. Automatica, 11: 3-24.

Hames, R. J., D. F. Merrion, and H. S. Ford. 1971. Some effects of fuel injection system on diesel exhaust emissions. SAE Paper No. 710671.

Harris, S. L. and D. A. Mellichamp. 1980. On-line identification of process dynamics: Use of multifrequency binary sequences. Ind. Eng. Chem. Process. Des. Dev., 19: 166-174.

Henein, N. A. 1972. Combustion and Emission formation in fuel sprays injected into swirling air. SAE Paper No. 720102.

Henein, N. A. 1979. Analysis of pollutant formation and control and fuel economy in diesel engines. 283-325. In N. A. Chigier Ed., Energy and combustion science. Pergamon, London.

Hiroyasu, H., M. Arai, and K. Nakanishi. 1980. Soot formation and oxidation in diesel engines. SAE Paper No. 800252.

Hsia, T. C. 1977. System identification: Least-squares methods. Lexington Books, Lexington, MA.

Khan, I. M., Wang, C. H. T., and B. E. Langridge. 1972. Effect of air swirl on smoke and gaseous emissions from direct-injection diesel engines. SAE Paper 720102.

Kittleson, D. B., D. F. Dolan, and J. A. Verraut. 1978. Investigation of a diesel exhaust aerosol. SAE Paper No. 780109.

Koppel, Lowell B. 1968. Introduction to control theory. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Lindgren, B. W. 1976. Statistics theory. 3rd Ed. Macmillan, New York.

Ljung, L. 1987. System identification: theory for the user. Prentice-Hall, Englewood Cliffs, NJ.

Ljung, L., and K. Glover. 1981. Frequency domain vs. time domain methods in system identification. Automatica, 17, No. 1: 71-86.

Lyn, W. -T. 1970. The spectrum of diesel combustion research. Proc. of Instn. Mech. Engrs., 184, Pt. (35)1.

Mehra, R. K. 1981. Choice of input signals. In P. Eykhoff Ed., Trends and progress in system identification. Pergamon Press, New York.

Morris, R. L., R. H. Borcherts, M. V. Walick, and H. G. Hopkins. 1982. Spark ignition engine model building-An identification approach to throttle-torque response. Int. J. of Vehicle Design, 3, No. 1: 48-60.

Nieman, R. E., D. G. Fisher and D. E. Seborg. 1971. A review of process identification and parameter estimation techniques. Int. J. Control, 13: 209-264.

Norris-Jones, S. R., T. Hollis, and C. N. F. Waterhouse. 1985. A study of the formation of particulates in the cylinder of a direct injection diesel engine. Paper No. 840419. SAE Trans., 93, sec. 3: 150.

Obert, E. F. 1973. Internal combustion engines and air pollution. Harper & Row, Publishers, Inc., New York.

Rake, H. 1980. Step response and frequency response methods. Automatica, 16: 519-526.

Ramirez, R. W. 1985. The FFT fundamentals and concepts. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Rao, C. R. 1973. Linear statistical inference and its applications. Wiley, New York.

Rissanen, J. 1985. Minimum description length principles. In S. Kotz and N. L. Johnson Ed., Encyclopedia of statistical sciences. Vol. V. Wiley, New York.

SAE J255a. 1978. Diesel engine smoke measurement. SAE information report. SAE Handbook. SAE, Warrendale, PA.

SAE J1157. 1976. Measurement procedure for evaluation of full-flow, light extinction smoke meter performance. SAE recommended practice. SAE Handbook. SAE, Warrendale, PA.

Sage, A. P., and J. L. Melsa. 1972. System identification. Academic Press, New York.

Sawaragi, Y., T. Soeda, and T. Nakamizo. 1981. Classical methods and time series estimation. In P. Eykhoff Ed., Trends and progress in system identification. Pergamon Press, New York.

Sawada, D. and T. Shigematsu. 1981. Improvement of spark ignition knock detector performance by learning control. SAE Publication SP-481:1-10.

Schwarzenbach, J. and K. F. Gill. 1984. System modelling and control. 2nd Ed. Edward Arnold Publishers Ltd., London.

Schweitzer, P. H. 1947. Must diesel engines smoke?. SAE Trans., 1, No. 3:476.

Shahed, S. M., W. S. Chiu, and W. T. Lyn. 1975. A mathematical model of diesel combustion. Conference on combustion in engines. Institution of Mechanical Engineers, Cranfield, England, Paper No. C94175.

Shroff, H. D. and D. Hodgetts. 1974. Simulation and optimization of thermodynamic processes of diesel engine. SAE Paper No. 740194.

Sweet, L. M. 1982. Automotive applications of modern control theory. Paper No. 820913. SAE Trans., 91, sec. 3:3022.

Van den Bos. 1967. Construction of binary multifrequency test signals. Identification in automatic control systems Part II. Preprints of the IFAC symposium. Praque, Czechoslovakia.

Van Gerpen, J. H. 1984. The effect of air swirl and fuel injection system parameters on diesel combustion. Ph.D. Dissertation. University of Wisconsin-Madison.

Van Gerpen J. H., C. -W. Huang, and G. L. Borman. 1986. The effects of swirl and injection parameters on diesel combustion and heat transfer. Paper No. 850265. SAE Trans., 94, sec. 2: 494,

Van Valkenburg, M. E. 1974. Network analysis. 3rd Ed. Prentice-Hall, Englewood Cliffs, NJ.

Veselinovic, B. 1985. Some more notes on the amount of diffusion burning causing smoke formation. SAE Paper No. 850050.

Vuk, C. T., M. A. Jones, and J. H. Johnson. 1976. The measurement and analysis of the physical character of diesel particulate emissions. Paper No. 760131. SAE Trans., 85: 556-597,

Wade, W. R. 1981. Light-duty diesel NOx-HC-Particulate trade-off studies. Paper No. 800335. SAE Trans., 89, sec. 2: 1379.

Walberg, B. and L. Ljung. 1986. Design variables for bias distribution in transfer function estimation. IEEE Trans. on Automatic Control, AC-31, No. 2: 134-144.

Walton, J. 1938. The fuel possibilities of vegetable oils. Gas and Oil Power, 33:167, 168.

Wellstead, P. E. 1981. Non-parametric methods of system identification. Automatica, 17, No. 1: 55-69.

Wellstead, P. E., and P. M. Zanker. 1981. Application of self-tuning to engine control. Ch. 12. In C. J. Harris and S. A. Billings, Ed. Self-tuning and adaptive control: Theory and applications. Peregrinus, London.

Wilson, R. P., Jr., E. B. Muir, and F. A. Pellicciotti. 1974. Emissions study of a single cylinder diesel engine. SAE Paper 740123.

Windett, G. P., and J. O. Flower. 1974. Sampled-data frequency response measurements of a large diesel engine. Int. J. Control, 19, No. 6: 1069-1086.

Yu, R. C., and S. M. Shahed. 1982. Effects of injection timing and exhaust gas recirculation on emissions from a D.I. diesel engine. Paper No. 811234. SAE Trans., 90, sec. 4: 3873.

Zadeh, L. A. 1962. From circuit theory to system theory. Proc. IRE 50: 856-865.

Zhang, N., J. V. Perampral, and R. K. Byler. 1985. Automatic control system for optimizing diesel engine performance. ASAE Paper No. 85-1582.

ACKNOWLEDGEMENT

The author wishes to express his deep gratitude to Dr. Richard J. Smith for his unselfish guidance and encouragement in the course of his research.

The author would also like to express his sincere appreciation to Dr. Stephen J. Marley, co-major professor and interim head of the Agricultural Engineering Department, who provided him with financial support, counsel and encouragement.

He is also appreciative to Dr. Jon H. Van Gerpen, a graduate committee member, who gave valuable lectures regarding internal combustion engines and very helpful suggestions during this study.

Special thanks are due to Professor Morton M. Boyd for his helpful ideas, encouragement and interest in this research as a graduate committee member.

Sincere appreciation is also given to Dr. Loren W. Zachary for serving as the minor representative on the graduate committee in Engineering Science and Mechanics.

He wishes to acknowledge Dr. Carl J. Bern and Dr. Lawrence A. Johnson for their willingness to serve on the graduate committee.

He would like to thank Mr. Harold H. Mesenbrink, department workshop supervisor, and Mr. Milton J. Moyer, electronic technician, for their technical help in preparing the engine set-up.

He also wishes to thank graduate colleagues especially Munir A. Sargana and Amjad P. Chaudhary, and visiting professor from India, Pawan K. Gupta, for their help while performing the engine experiment.

The author is indebted to the Agricultural Engineering Department for providing resources and facilities for his research.

His deepest gratitude is expressed to his parents for their love, encouragement and patience.

Finally to his wife, Bo-Kyung, he would like to express his gratitude for her encouragement and for taking care of their lovely children, and to his sons, Young-Sang and Min-Sang, for their patience, endless support and love.

APPENDIX A: FREQUENCY RESPONSE GRAPHS IN SIMULATION STUDY

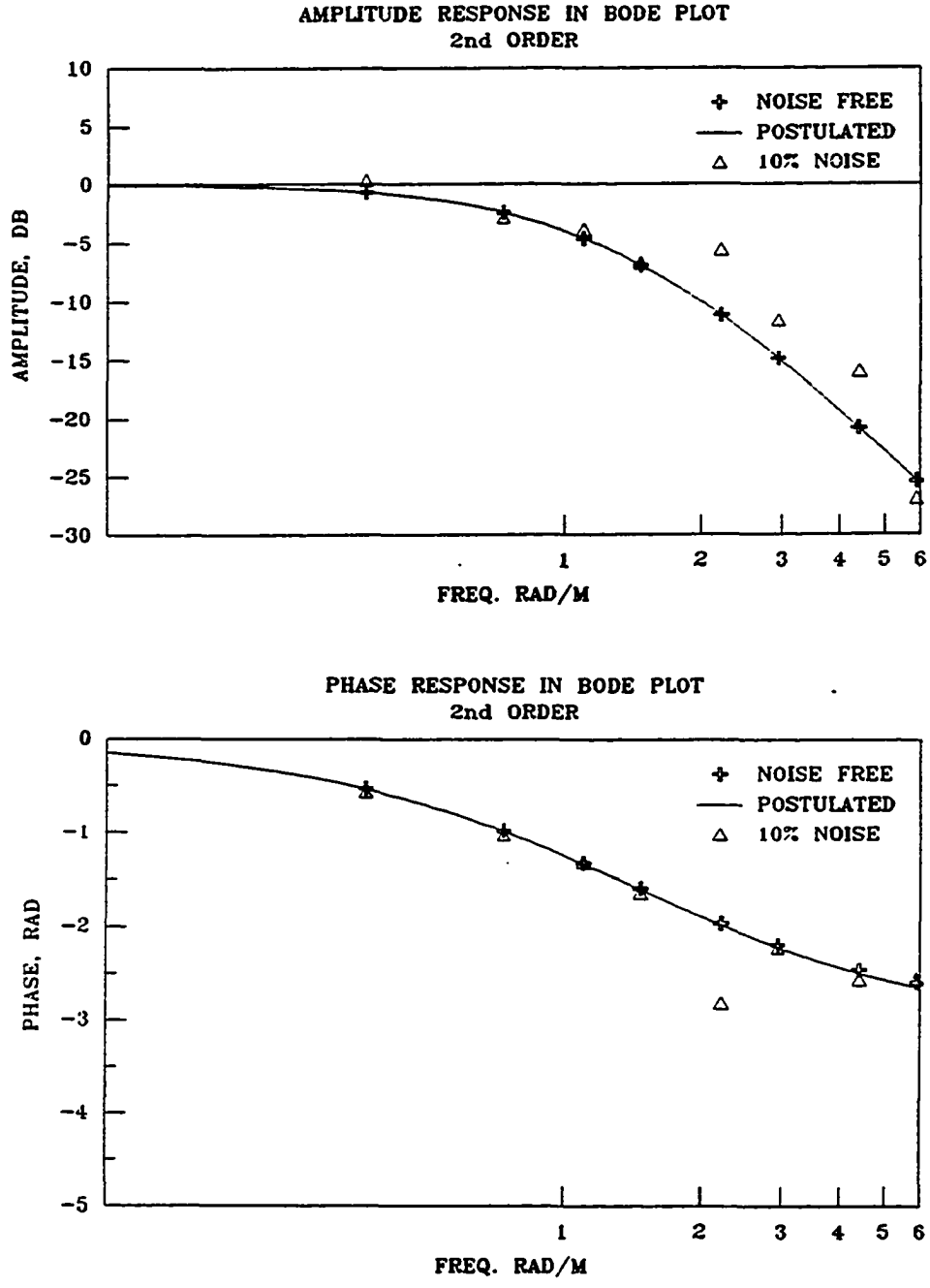
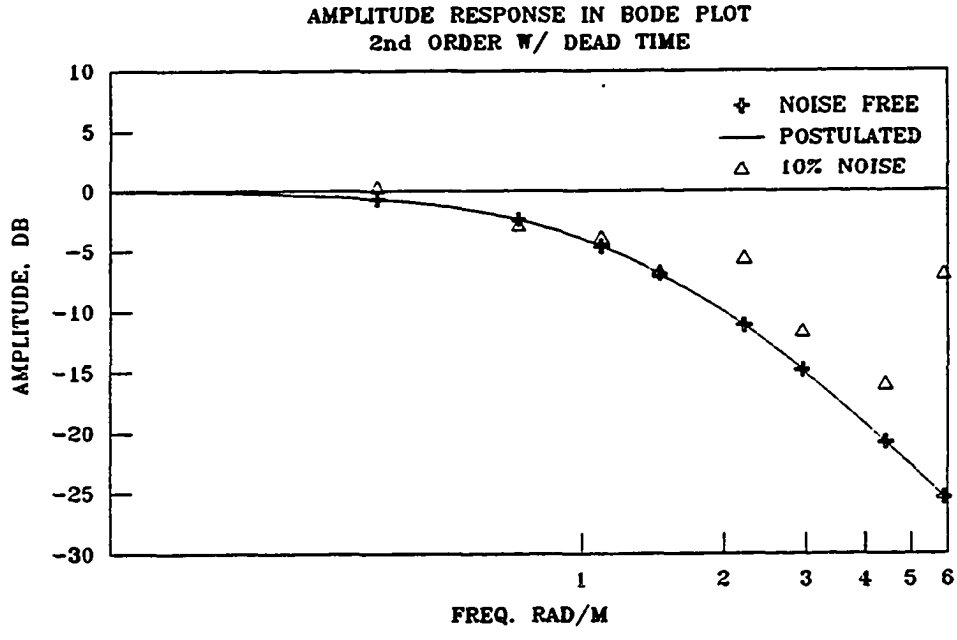


FIGURE A.1. Frequency response plot of 2nd order system identification



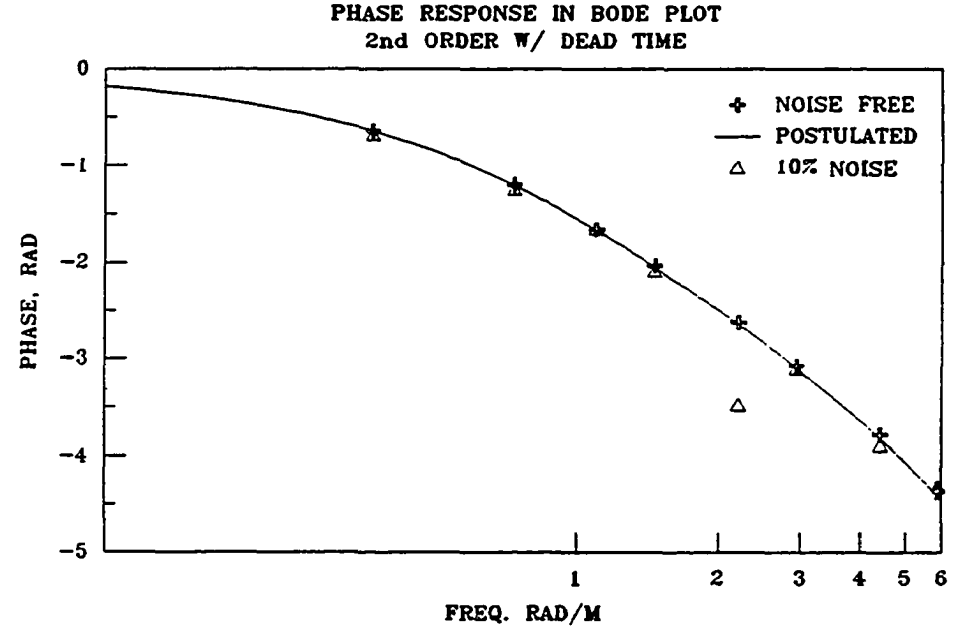


FIGURE A.2. Frequency response plot of 2nd order w/dead time identification

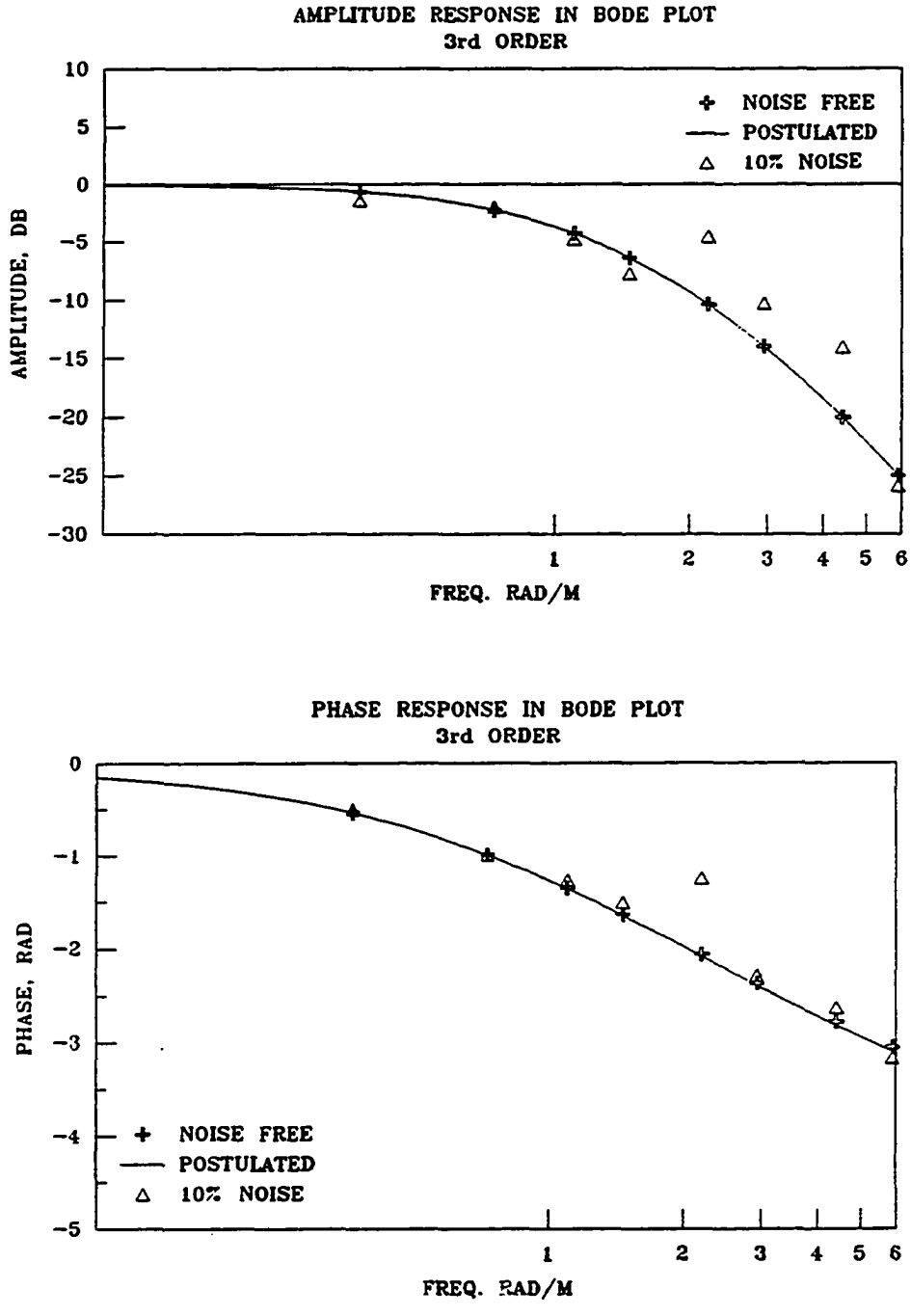
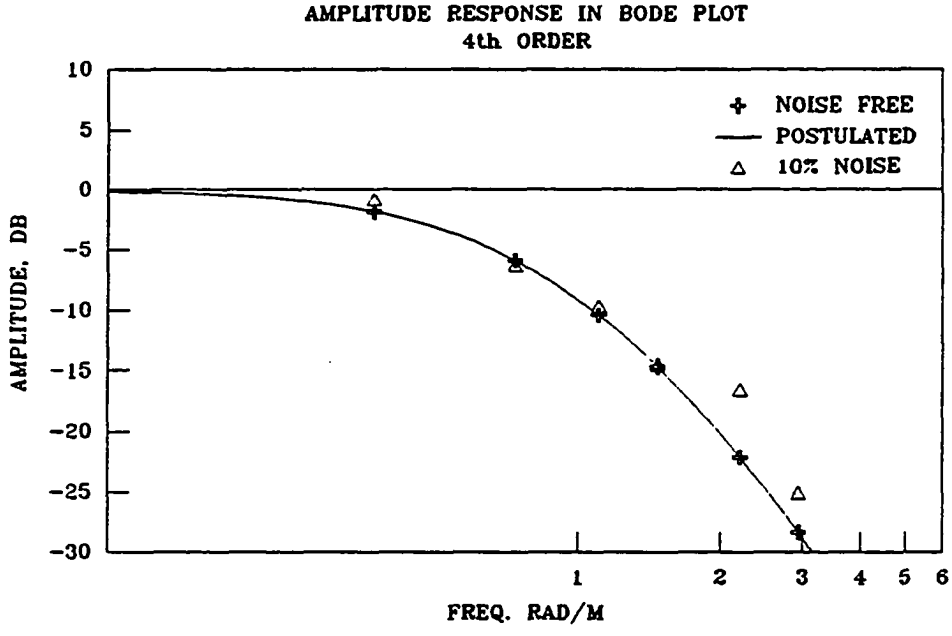


FIGURE A.3. Frequency response plot of 3rd order system identification



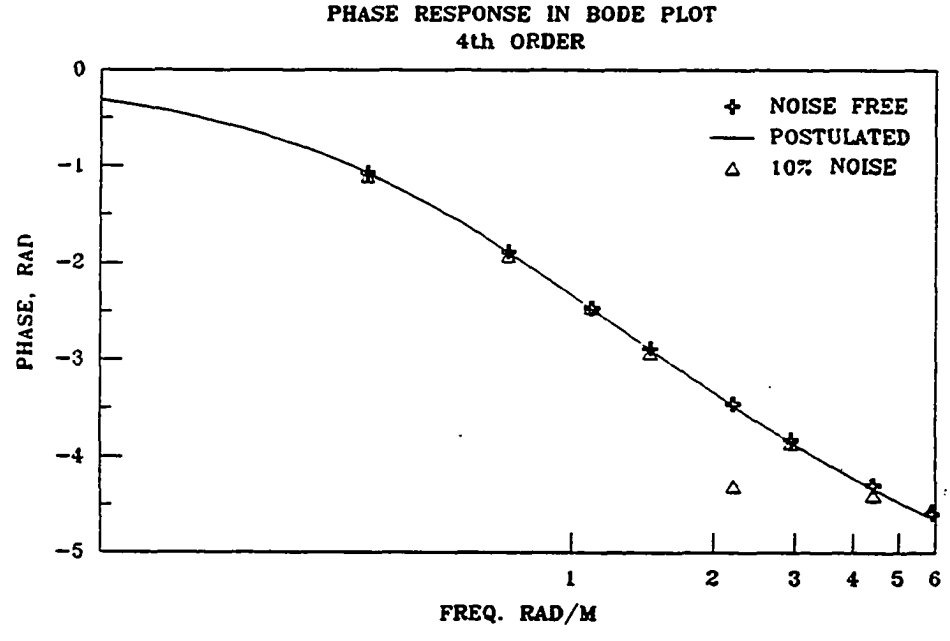


FIGURE A.4. Frequency response plot of 4th order system identification

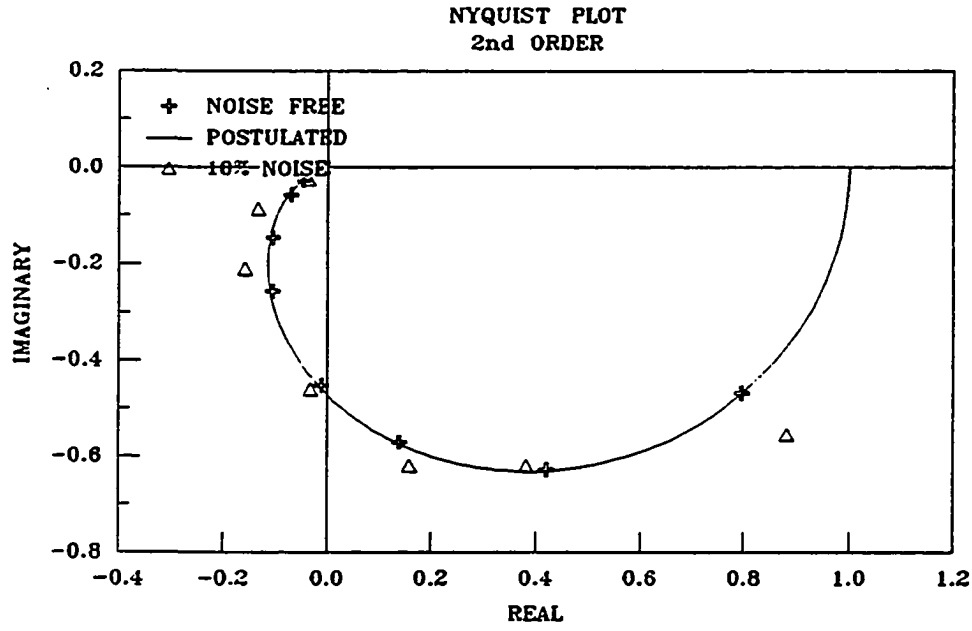


FIGURE A.5. Nyquist plot of 2nd order system identification

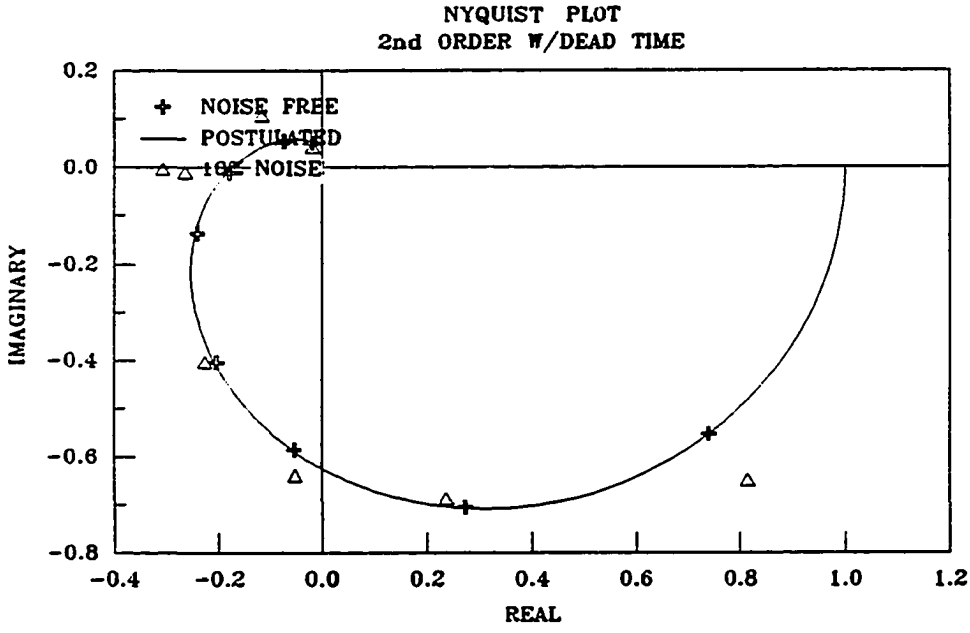


FIGURE A.6. Nyquist plot of 2nd order w/dead time identification

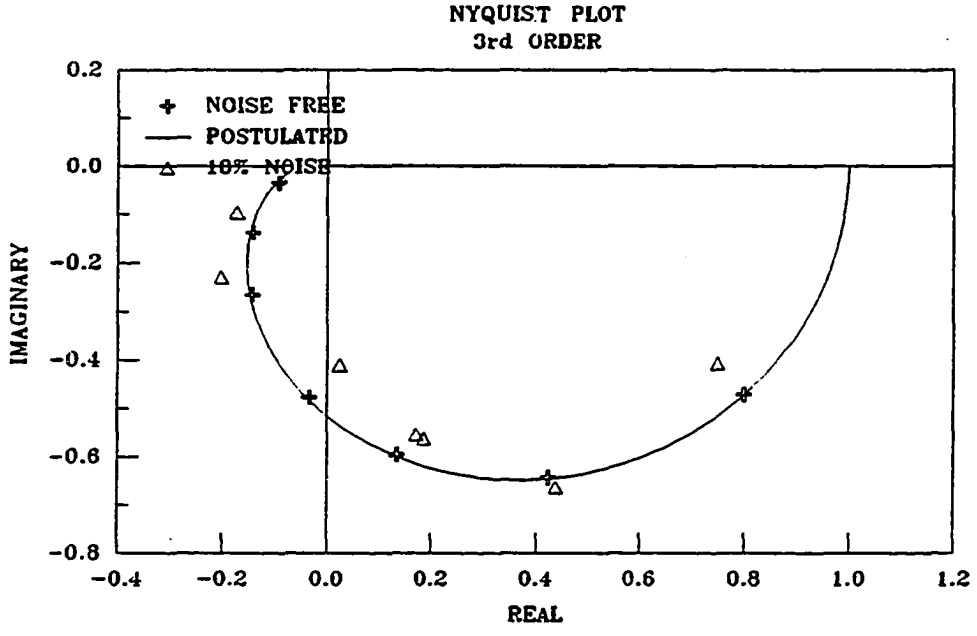


FIGURE A.7. Nyquist plot of 3rd order system identification

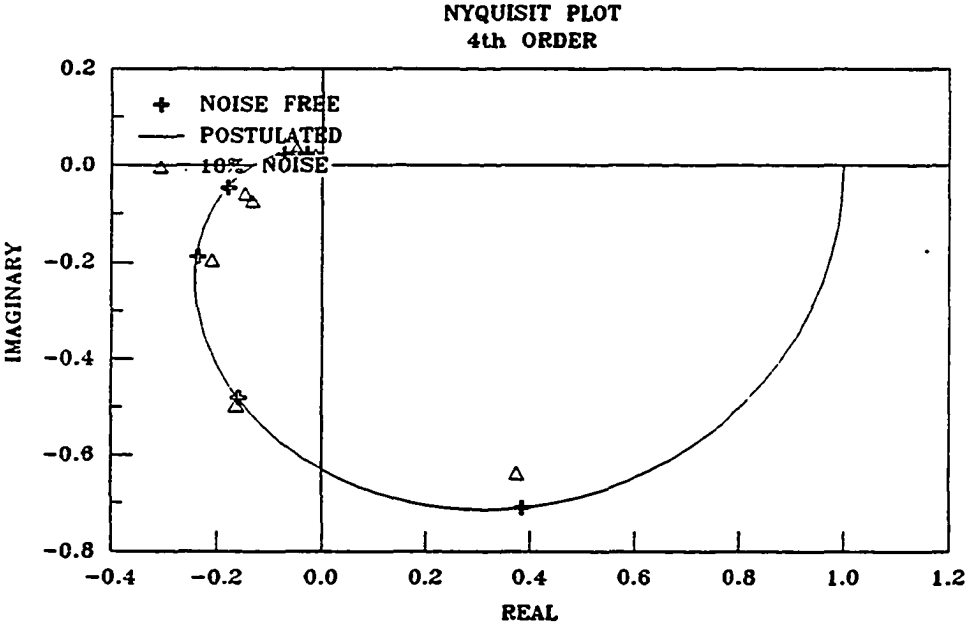


FIGURE A.8. Nyquist plot of 4th order system identification

APPENDIX B: SMOKE RESPONSE GRAPHS IN TIME DOMAIN

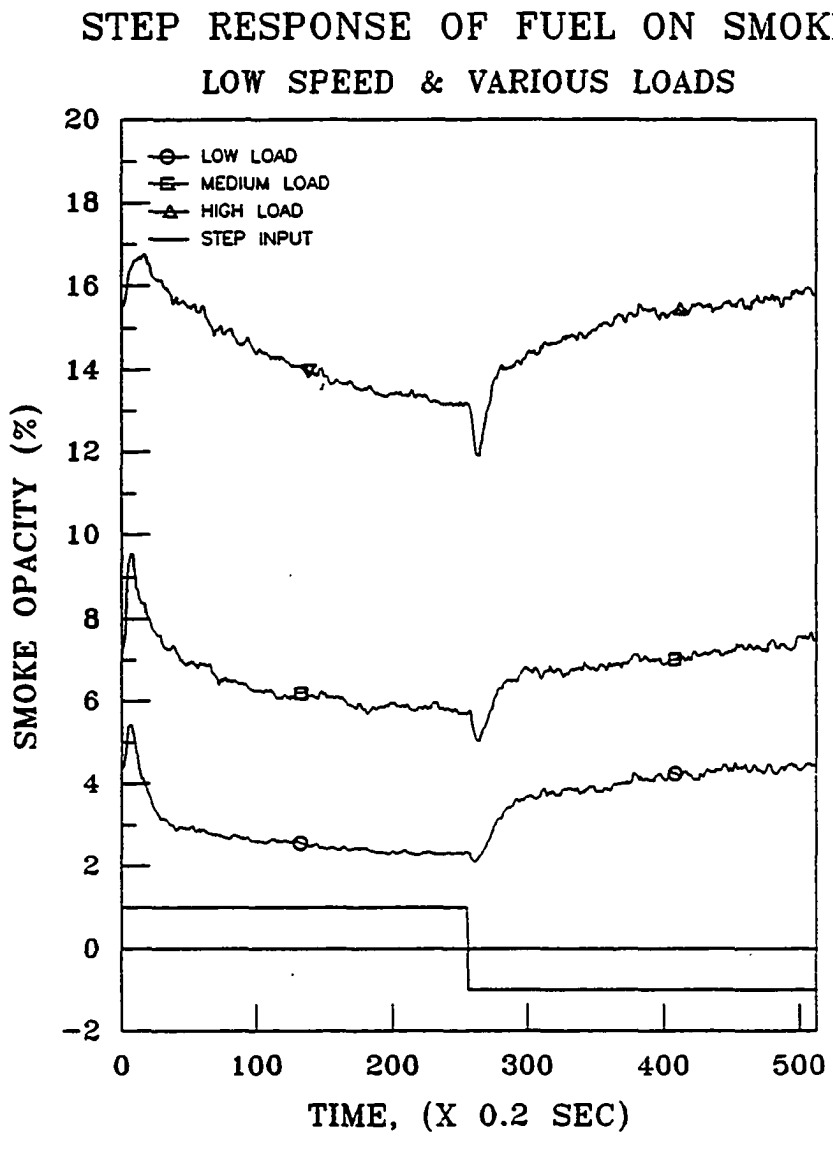


FIGURE B.1. Step response of fuel change on smoke opacity at 1000 r/min

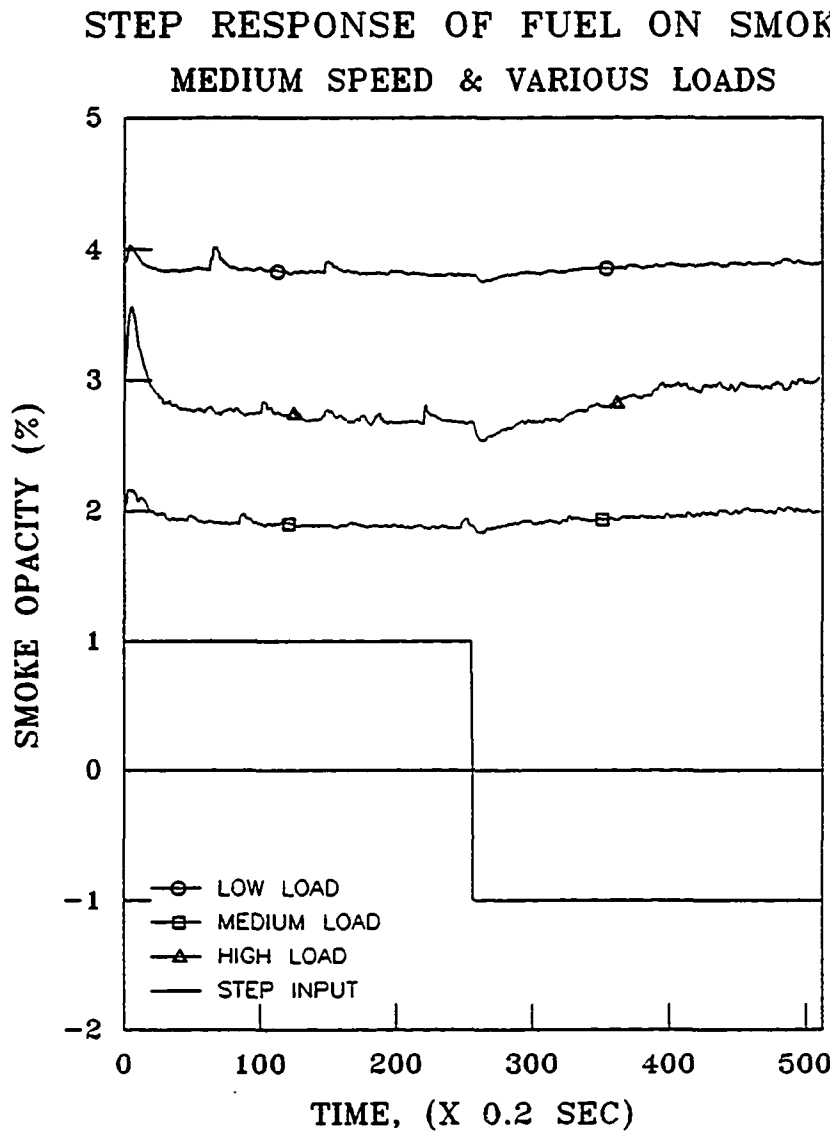


FIGURE B.2. Step response of fuel change on smoke opacity at 1500 r/min

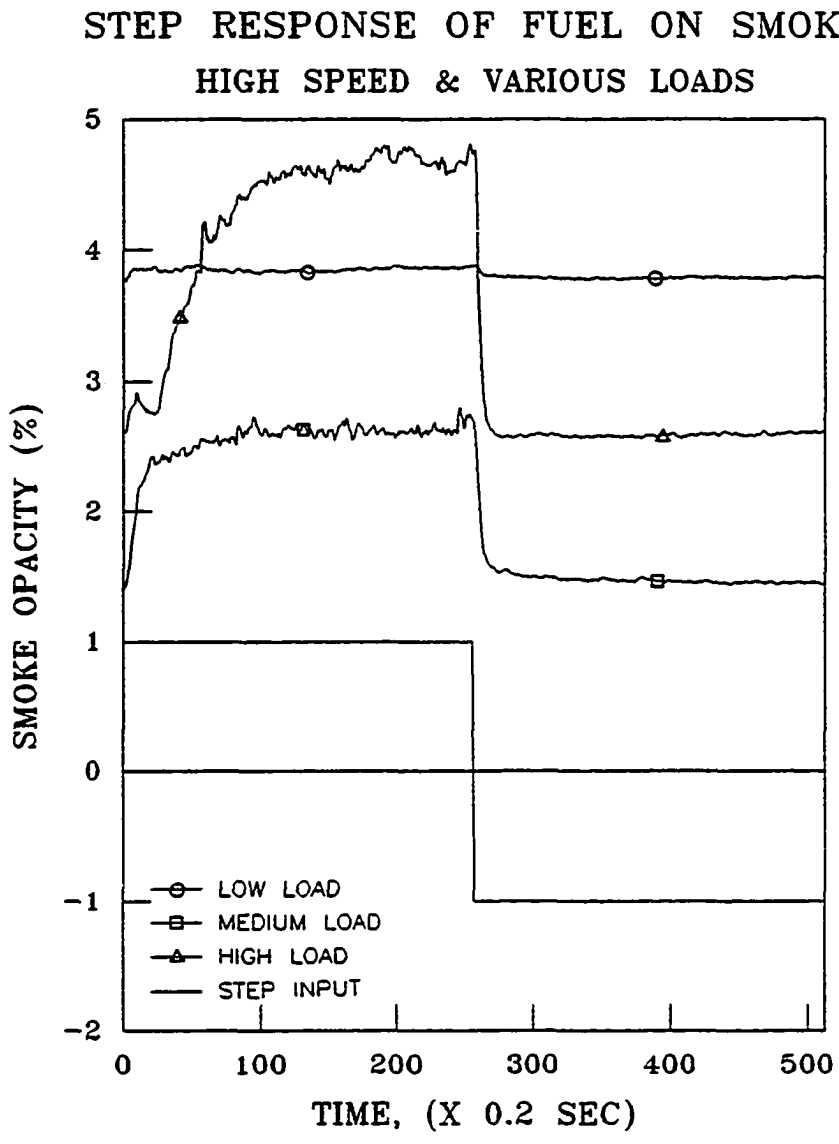


FIGURE B.3. Step response of fuel change on smoke opacity at 2000 r/min

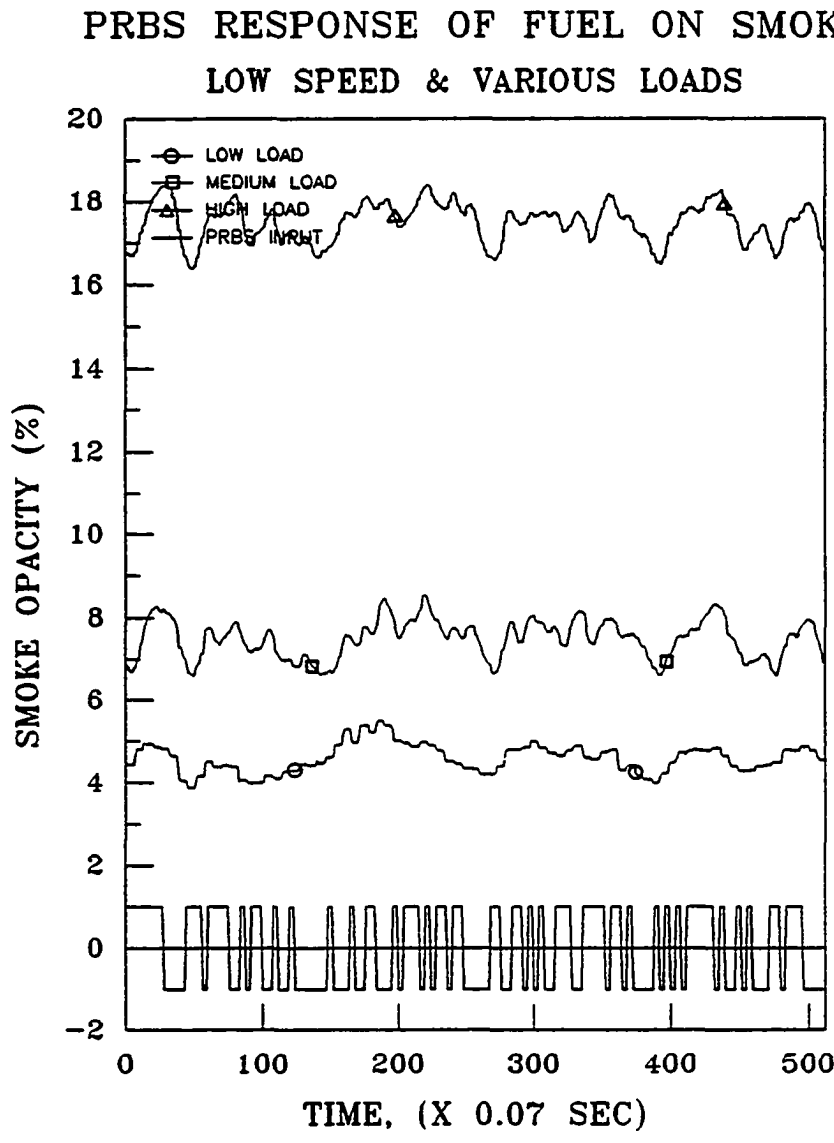


FIGURE B.4. PRBS response of fuel change on smoke opacity at 1000 r/min

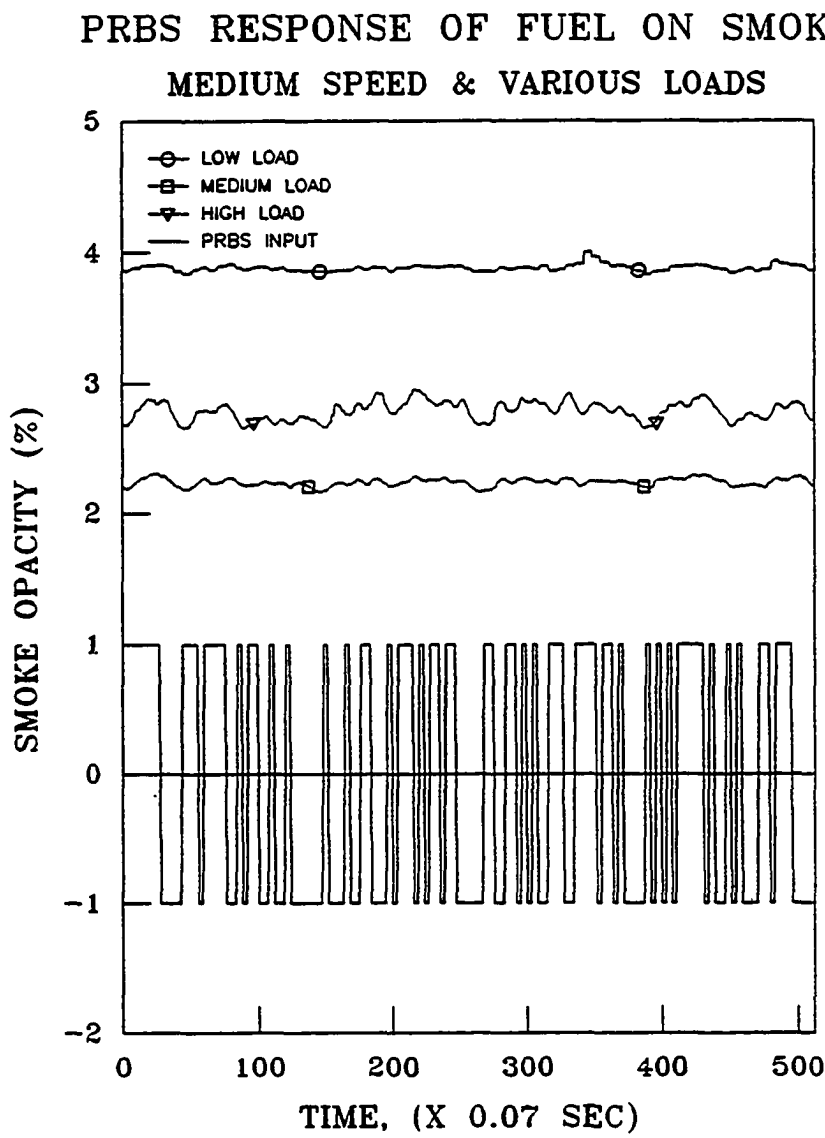


FIGURE B.5. PRBS response of fuel change on smoke opacity at 1500 r/min

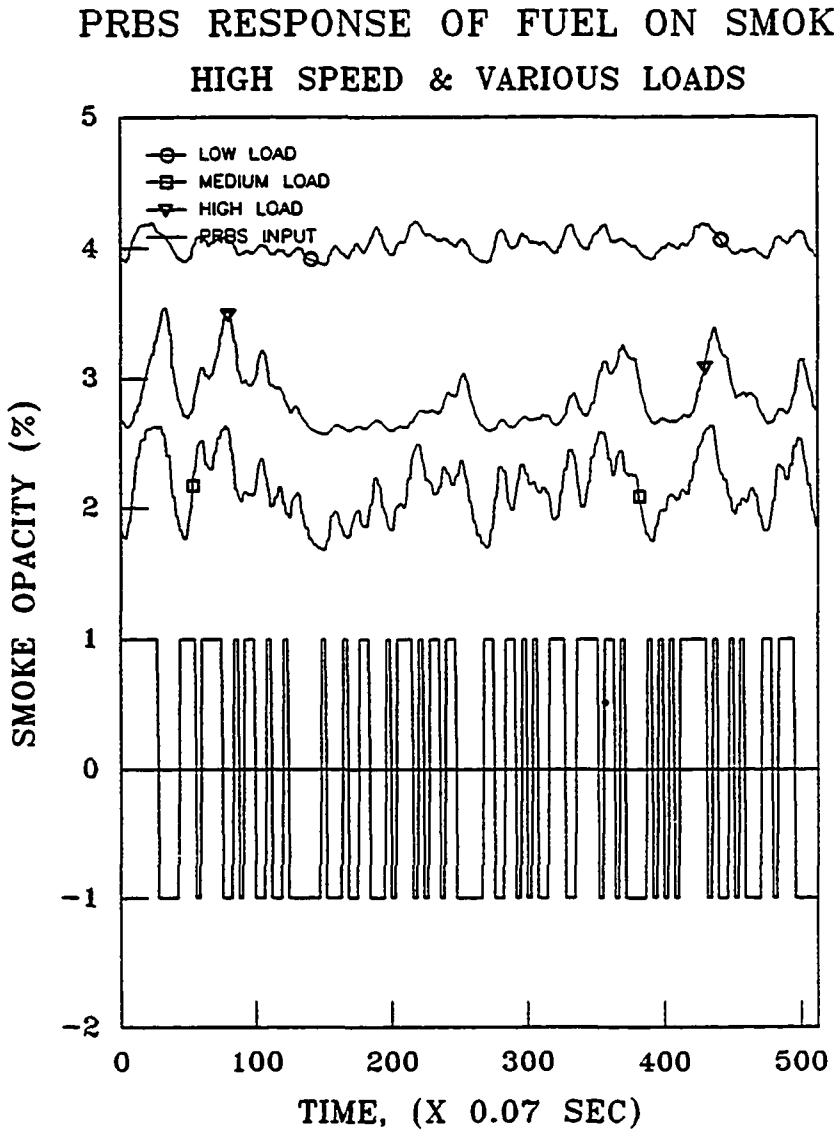


FIGURE B.6. PRBS response of fuel change on smoke opacity at 2000 r/min

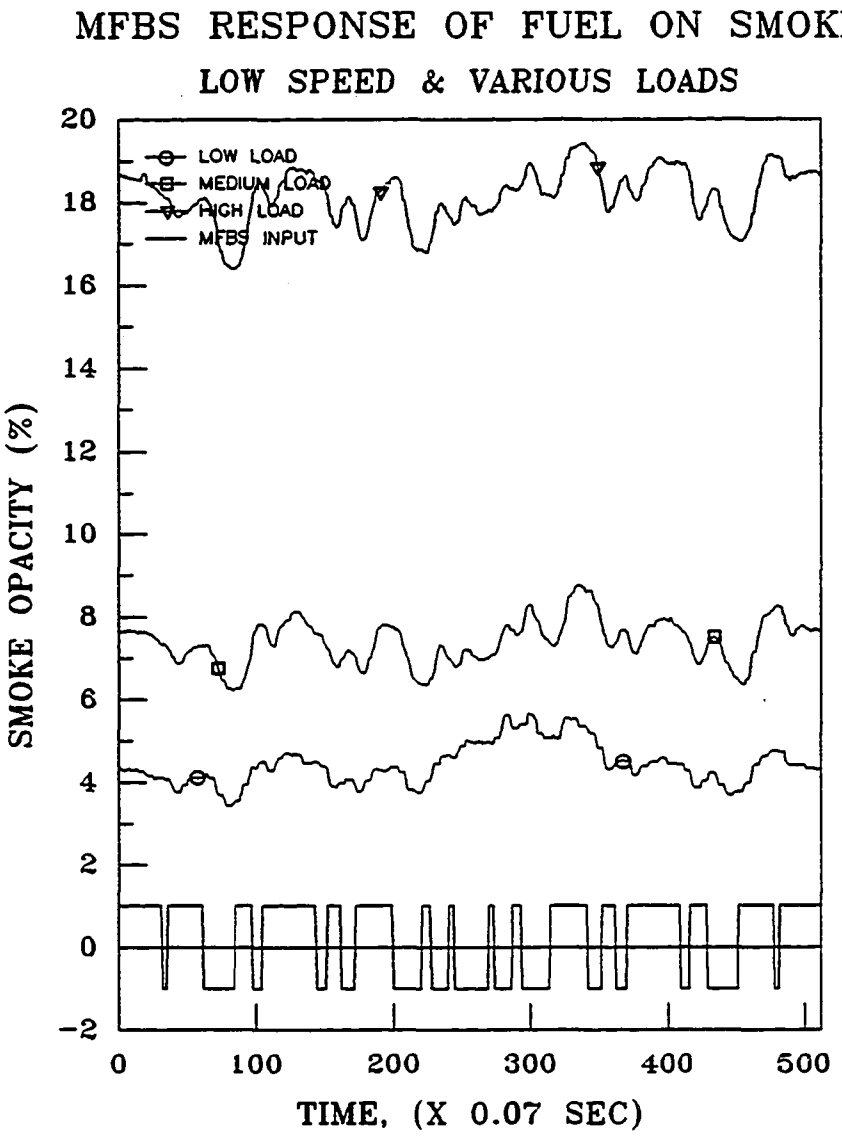


FIGURE B.7. MFBS response of fuel change on smoke opacity at 1000 r/min

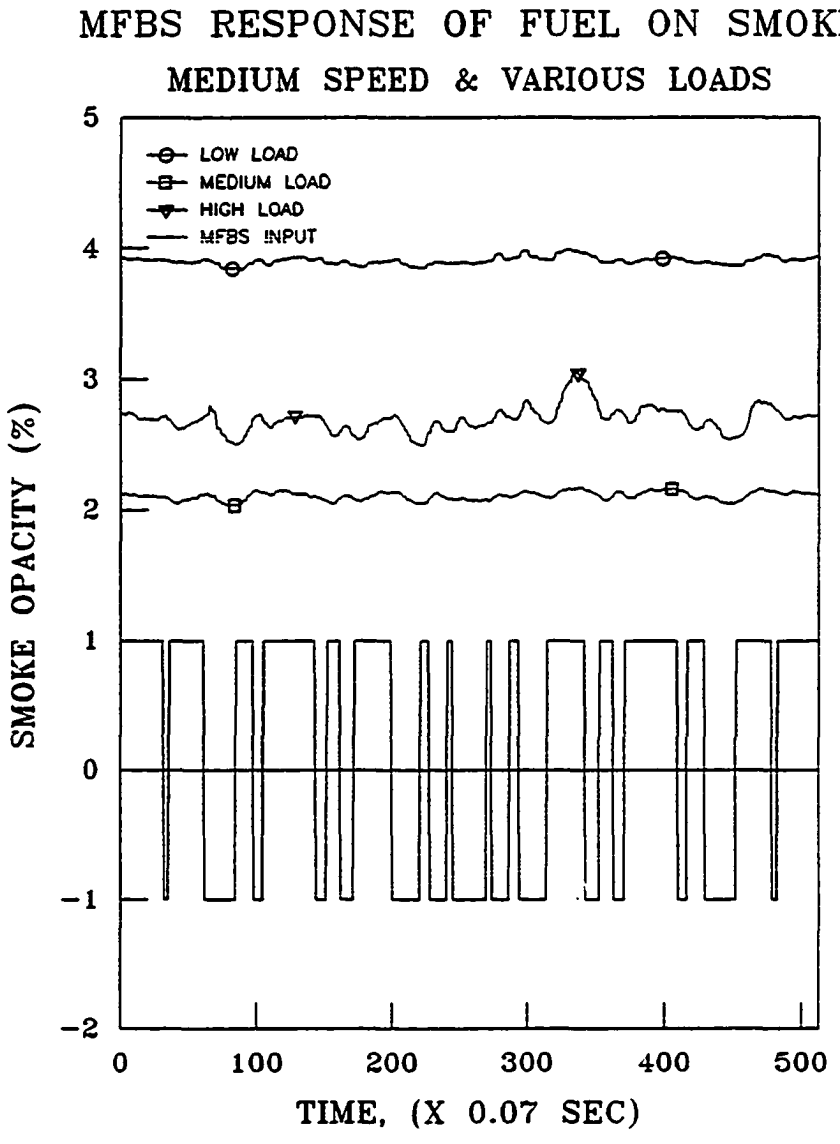


FIGURE B.8. MFBS response of fuel change on smoke opacity at 1500 r/min

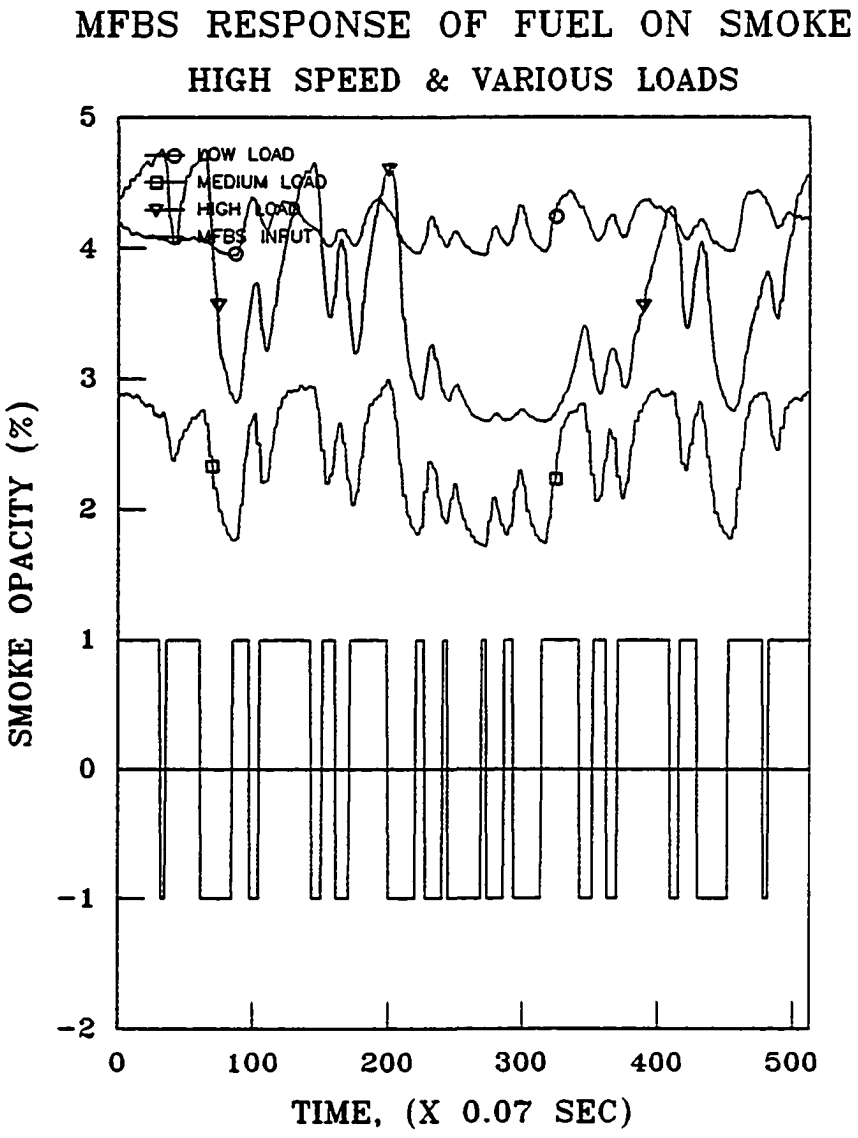


FIGURE B.9. MFBS response of fuel change on smoke opacity at 2000 r/min

APPENDIX C: FREQUENCY RESPONSE GRAPHS IN ENGINE EXPERIMENT

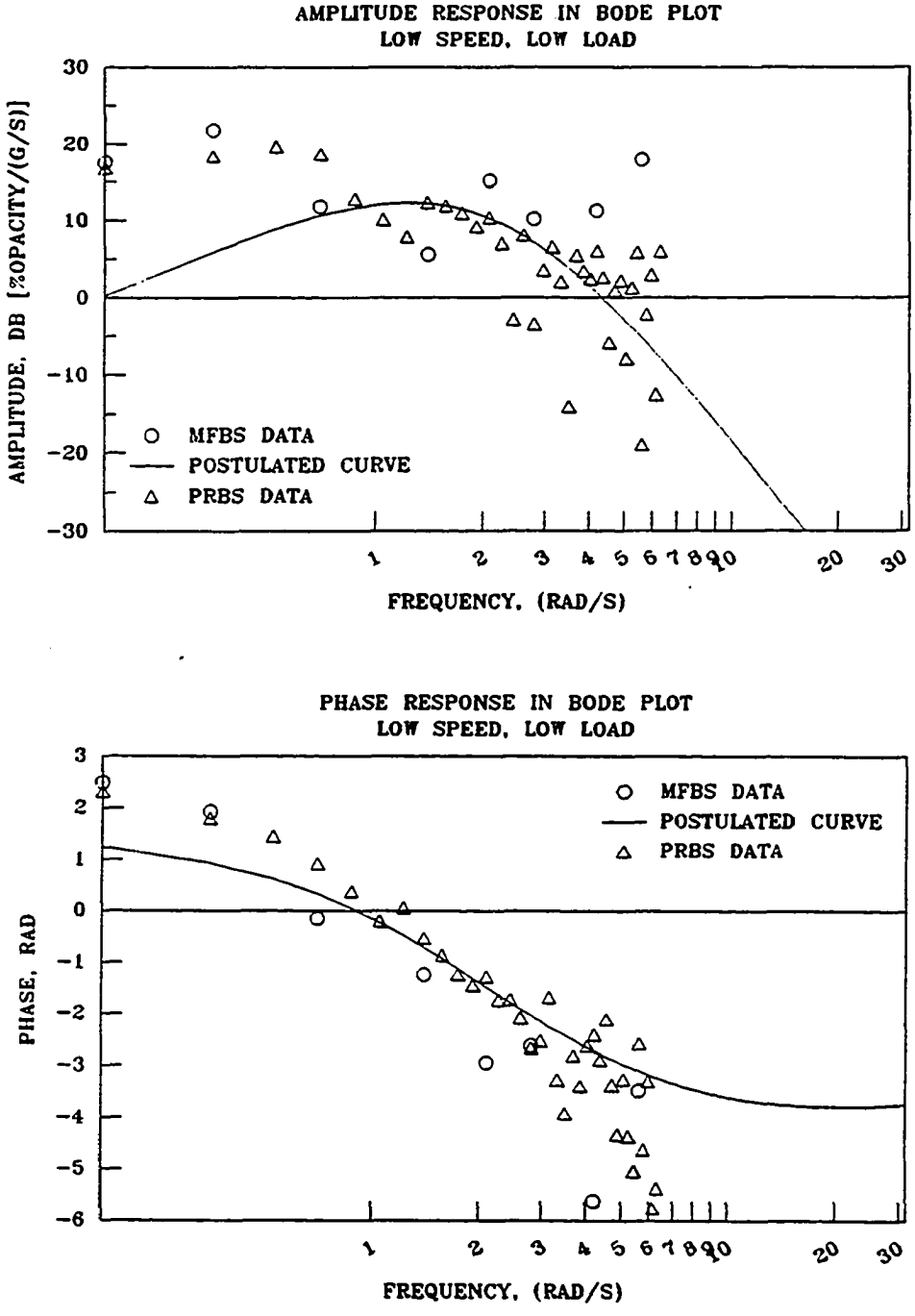


FIGURE C.1. Amplitude and phase angle in transfer function for experimental data and postulated curve at low speed and low load

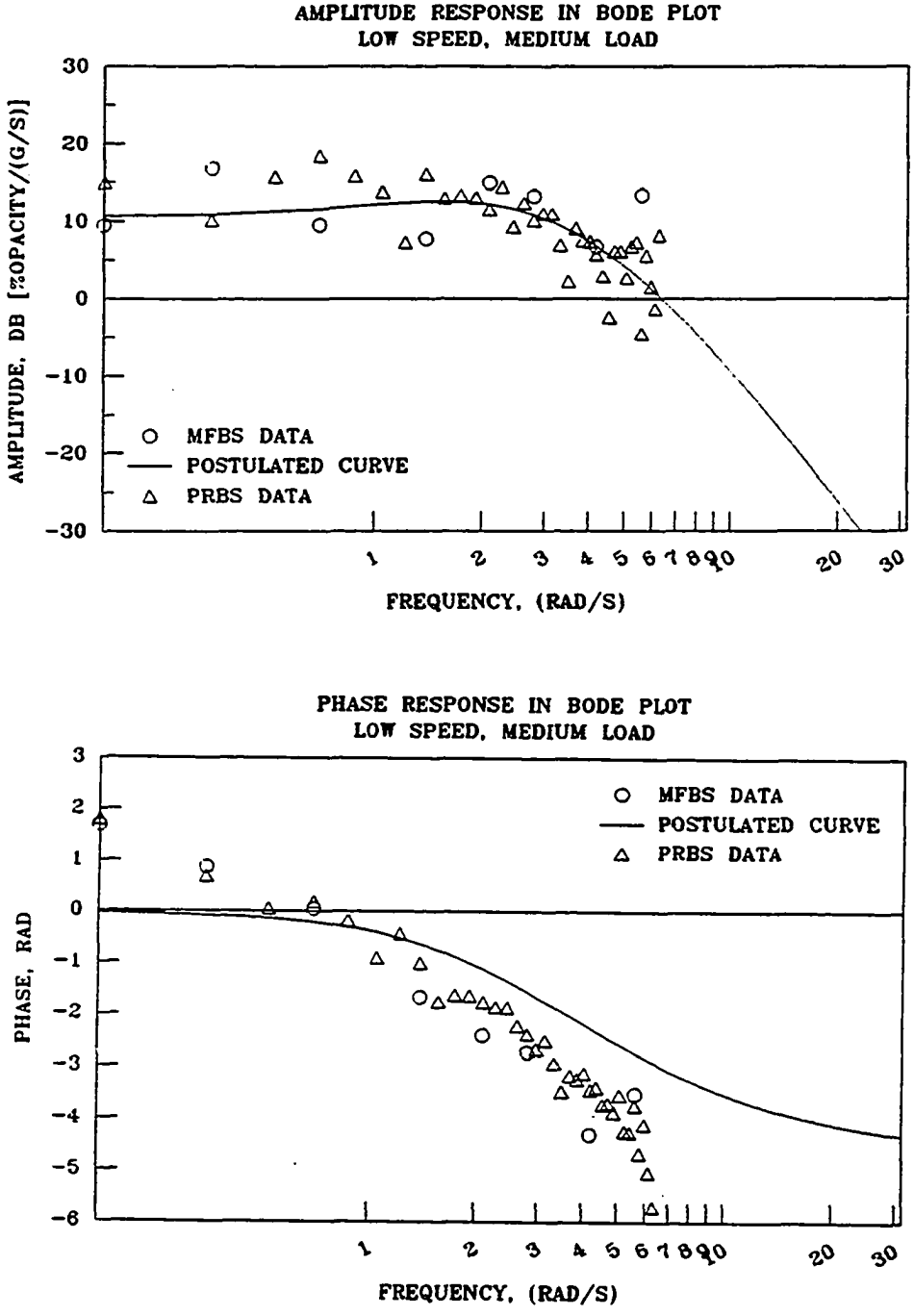
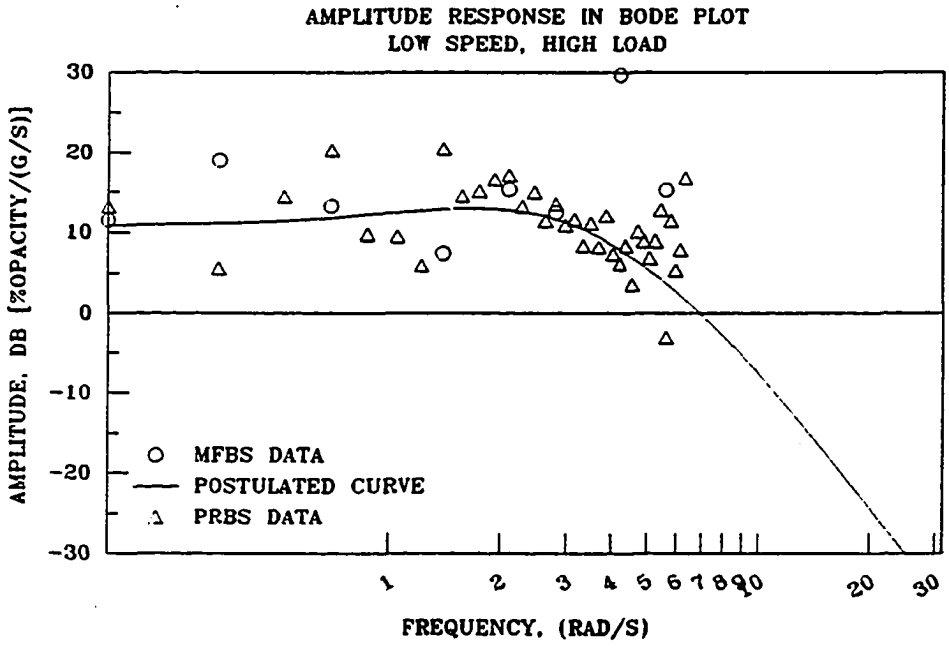


FIGURE C.2. Amplitude and phase angle in transfer function for experimental data and postulated curve at low speed and medium load



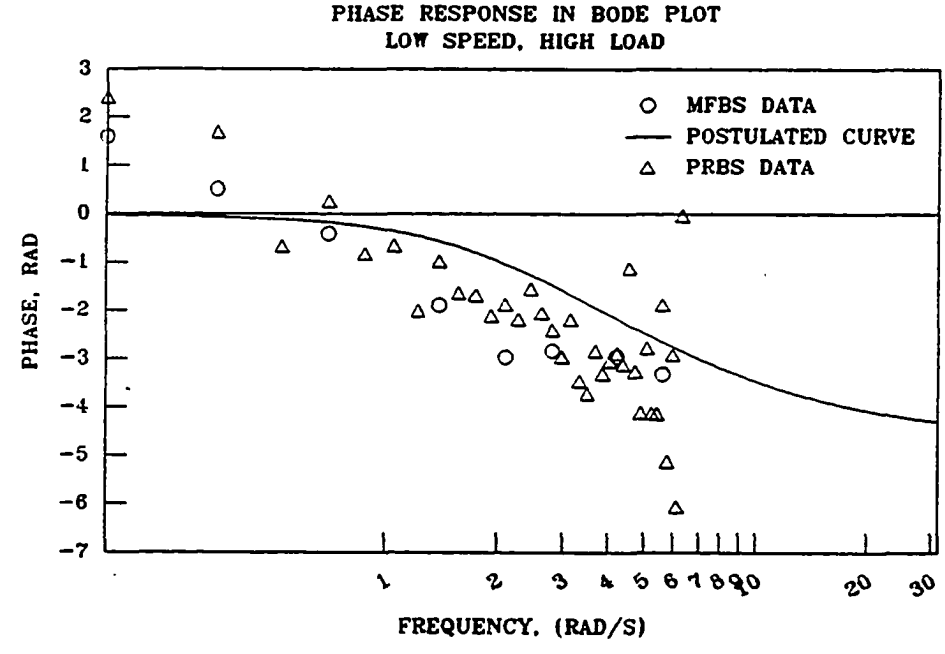
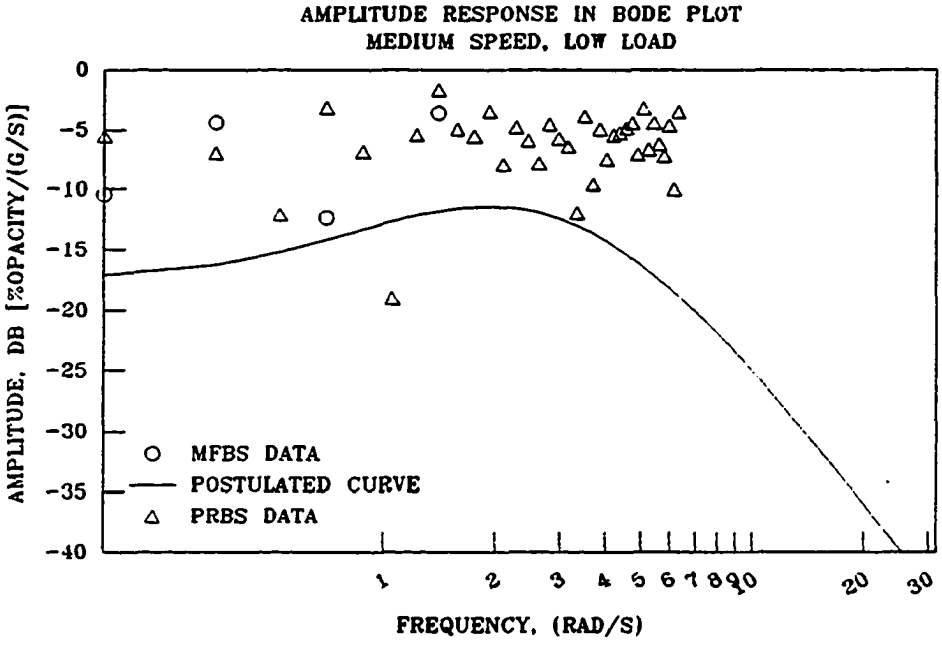


FIGURE C.3. Amplitude and phase angle in transfer function for experimental data and postulated curve at low speed and high load



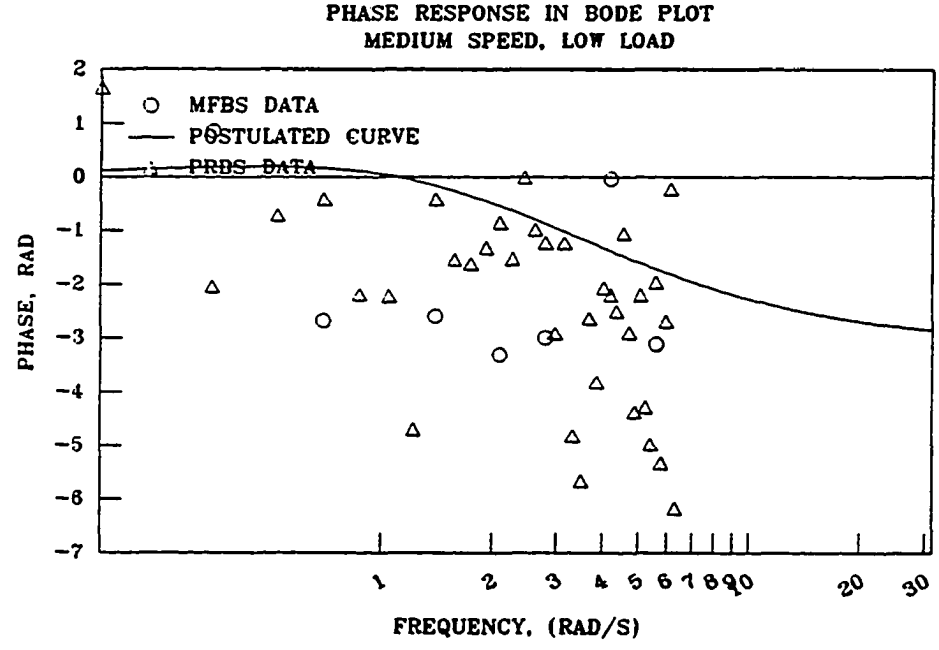
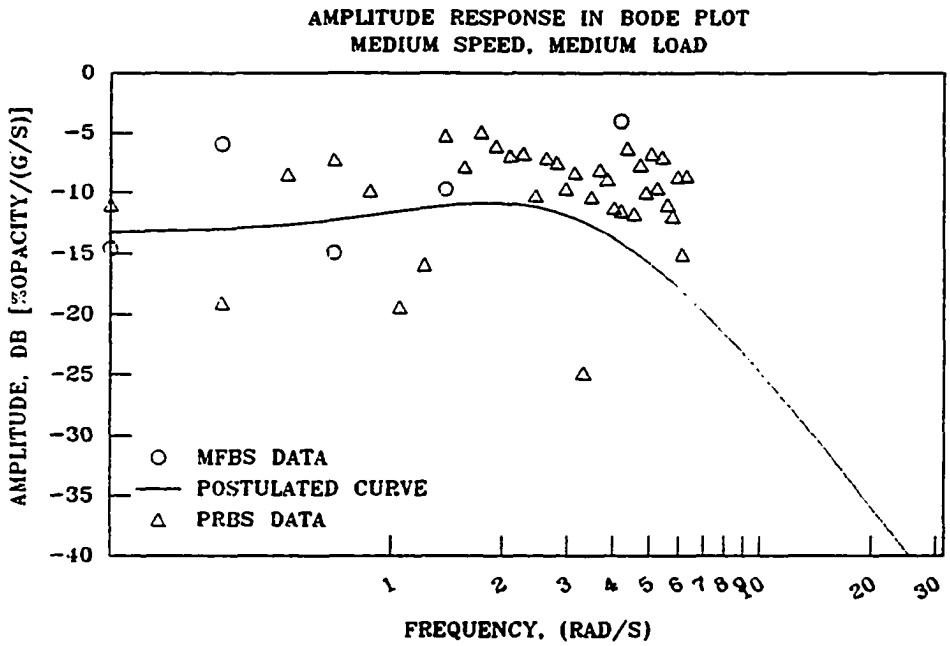


FIGURE C.4. Amplitude and phase angle in transfer function for experimental data and postulated curve at medium speed and low load



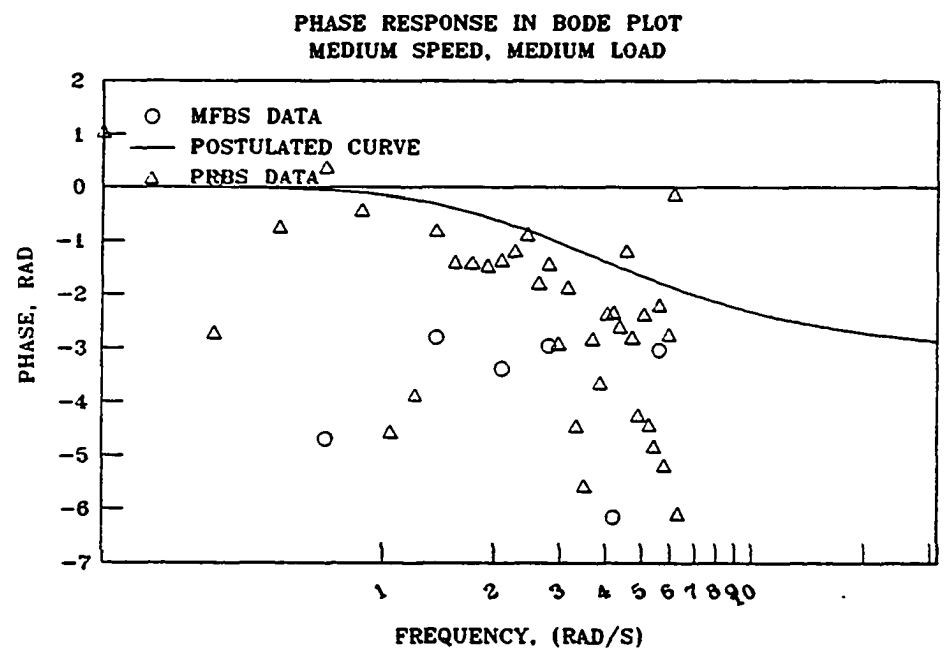
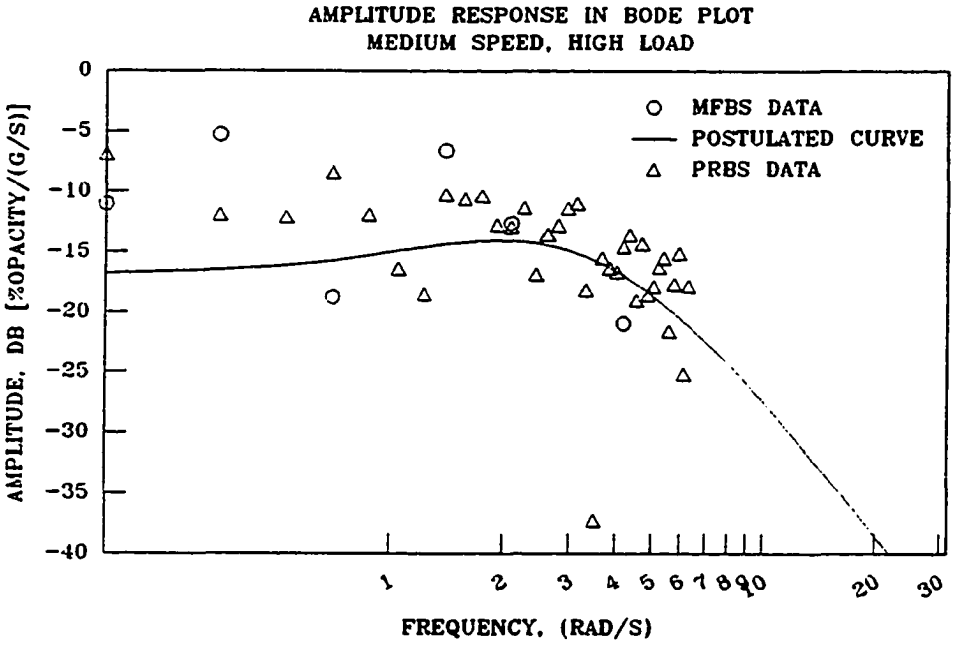


FIGURE C.5. Amplitude and phase angle in transfer function for experimental data and postulated curve at medium speed and medium load



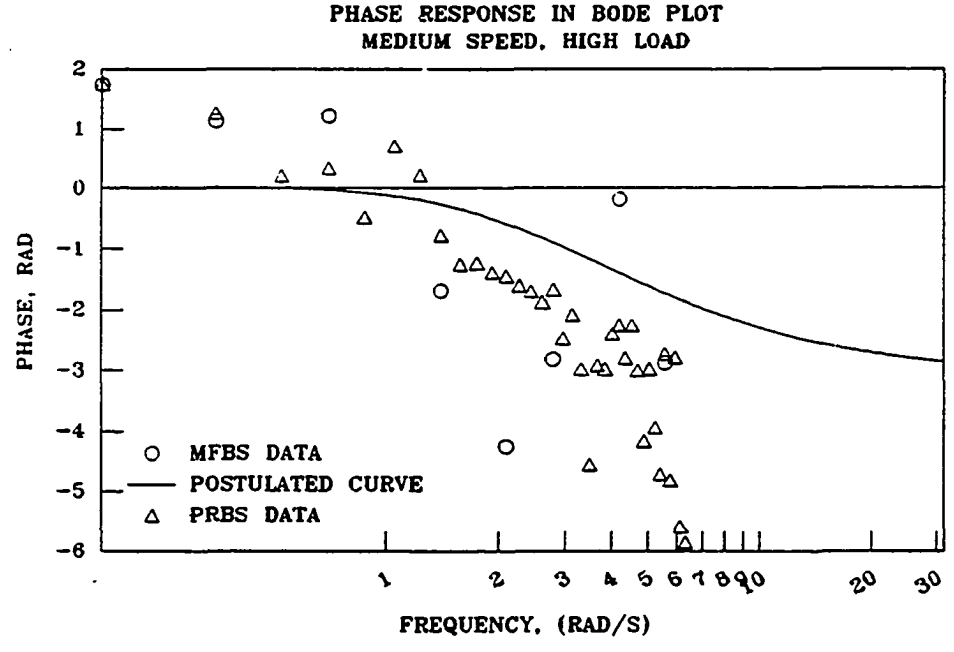
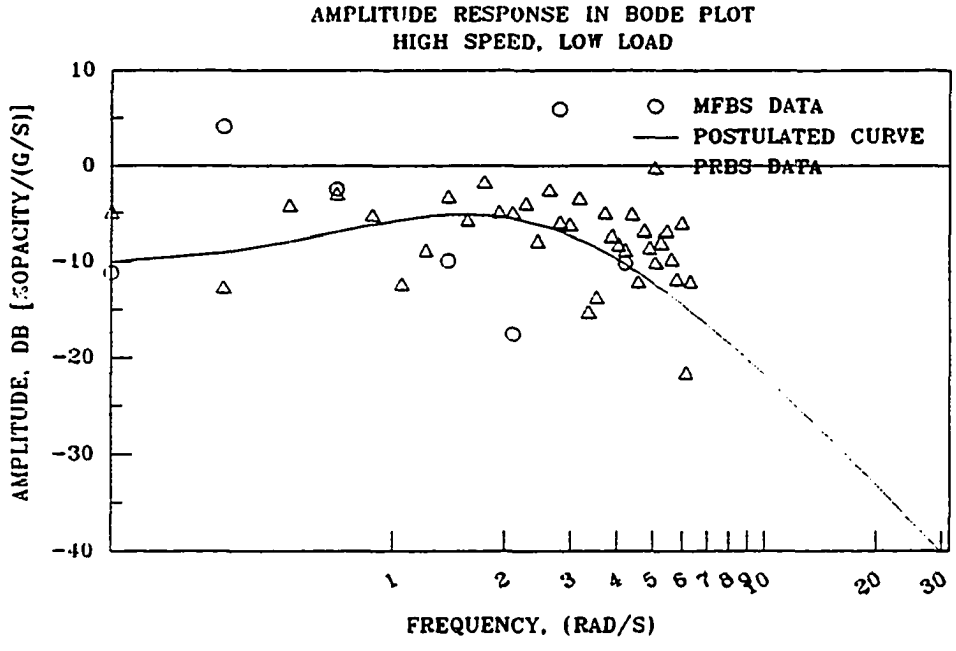


FIGURE C.6. Amplitude and phase angle in transfer function for experimental data and postulated curve at medium speed and high load



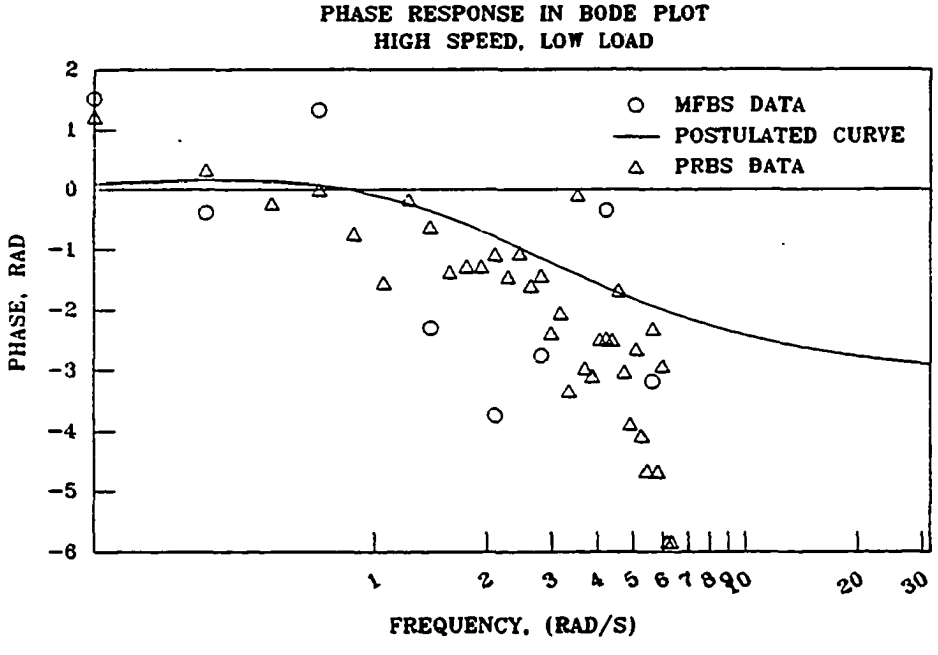


FIGURE C.7. Amplitude and phase angle in transfer function for experimental data and postulated curve at high speed and low load

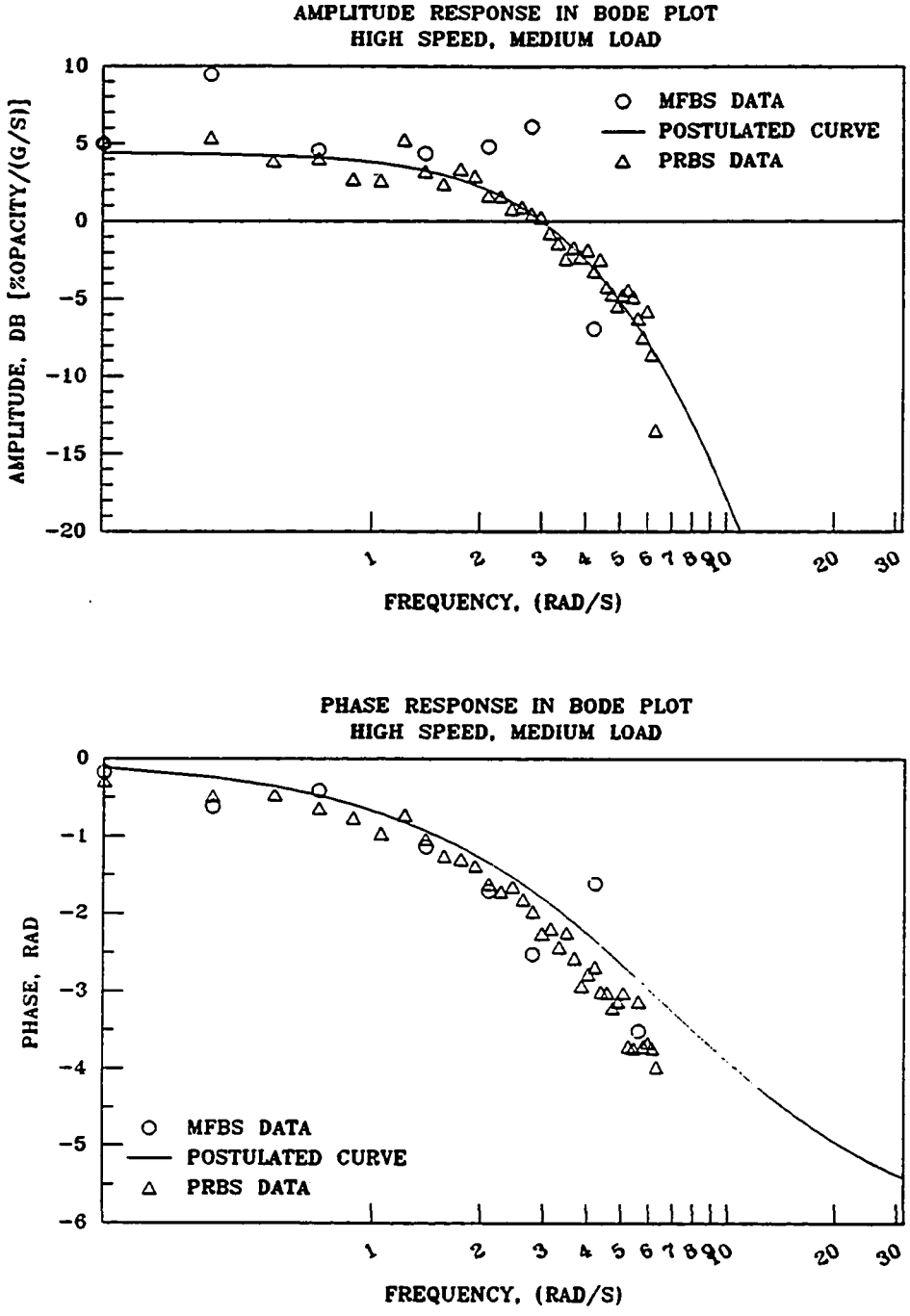
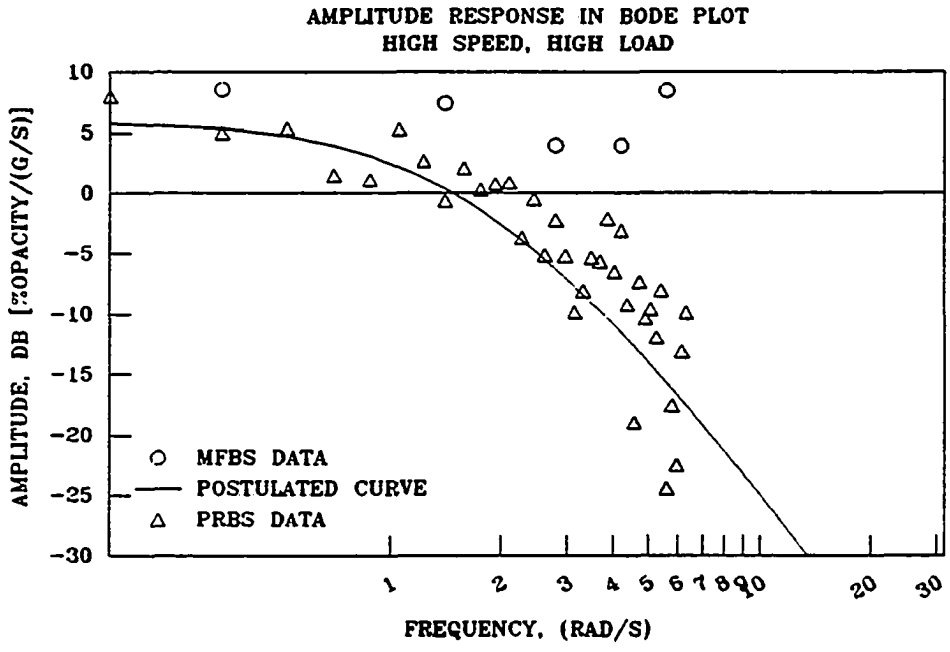


FIGURE C.8. Amplitude and phase angle in transfer function for experimental data and postulated curve at high speed and medium load



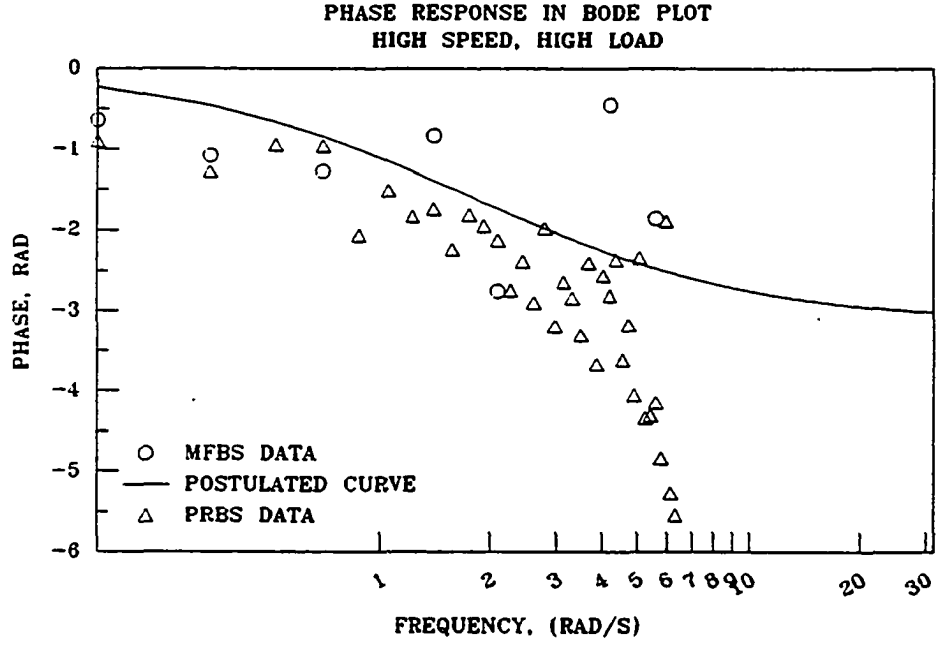


FIGURE C.9. Amplitude and phase angle in transfer function for experimental data and postulated curve at high speed and high load

APPENDIX D: PROGRAM LISTING FOR INPUT SIGNAL GENERATION

```
/*****    PRBS.C    ************************************************

    1. Program name
       PRBS.C

    2. Date
       September 1988

    3. Object
       To generate the pseudo random binary signal by Maximal
       length sequence

*********************************************************************/

#include <stdio.h>
#include <alloc.h>
#define P0 1                    /* Powers of 2 */
#define P1 2
#define P2 4
#define P3 8
#define P4 16
#define P5 32
#define P6 64
#define P7 128
#define P8 256
#define P9 512
#define P10 1024
#define P11 2047
int N,fb1,fb2,fb3,fb4,IB1,IB2,IB3,IB4;

main()
{
    int i,j,n,prbs[P11+1];
        int iii[P10+1],nhalf;
    FILE *fo;
    char filename[13];
    unsigned long *iseed;

    printf("\t\t***********************************************\n");
    printf("\t\t*                                             *\n");
    printf("\t\t*    PSEUDO RANDOM BINARY SIGNAL GENERATOR    *\n");
    printf("\t\t*                                             *\n");
    printf("\t\t*        (by Maximal length sequence)         *\n");
    printf("\t\t*                                             *\n");
    printf("\t\t*        H. K. Cho   September, 1988          *\n");
    printf("\t\t*                                             *\n");
    printf("\t\t*        1 < n < 12, N = (2**n) -1            *\n");
```

```
    printf("\t\t*                                                *\n");
    printf("\t\t************************************************\n");
        printf("\n\tEnter the file name for output : ");
        scanf("%s",filename);
        if( (fo = fopen(filename,"w")) == NULL){
                printf("Can't open file\n");
                exit(0);
        }
    printf("\n\tEnter the number of shift registers, n = ");
        scanf("%d",&n);
        while(n<2 || n>11) {
                printf("\tEnter the number again, 1<n<12, n = ");
                scanf("%d",&n);
        }
        iseed = (unsigned long*) malloc (sizeof(unsigned long));
        /** Total no. of signals, feed back registers **/
        switch(n){
                case  2: N= P2-1; fb1=0; IB1=P0; IB2= P1; break;
                case  3: N= P3-1; fb1=1; IB1=P1; IB2= P2; break;
                case  4: N= P4-1; fb1=2; IB1=P2; IB2= P3; break;
                case  5: N= P5-1; fb1=2; IB1=P2; IB2= P4; break;
                case  6: N= P6-1; fb1=4; IB1=P4; IB2= P5; break;
                case  7: N= P7-1; fb1=3; IB1=P3; IB2= P6; break;
                case  8: N= P8-1; fb1=1; fb3=2;fb4=3;
                                IB1=P1;IB2=P7;IB3=P2;IB4=P3;break;
                case  9: N= P9-1; fb1=4; IB1=P4; IB2= P8; break;
                case 10: N=P10-1; fb1=6; IB1=P6; IB2= P9; break;
                case 11: N=P11-1; fb1=8; IB1=P8; IB2=P10; break;
        }
        fb2 = n-1;
        *iseed = N;
        prbs[0] = 1;
        if(n!=8) {
            for(i=1;i<N;i++)
                prbs[i] = irbit(iseed);
        }
        else {
            for(i=1;i<N;i++)
                prbs[i] = irbit8(iseed);
        }
/*** Calculate index form ****************************/
        for(i=0;i<P10+1;i++)
                iii[i]=1;
    for(i=1,j=0;i<N;i++) {
            if(prbs[i-1] == prbs[i])
                    iii[j] ++;
        else
                    iii[++j]=1;
        }
        nhalf =(N+1)/2;
```

```
    if(prbs[0] == prbs[N-1]) iii[0] = iii[0]+iii[nhalf];
/*** Print on the screen ******************************/
        printf("\n\tPRBS signals with %d shift reg.\n\n",n);
        for(i=0;i<N;i++){
            printf("\t%4d = %d\n",i+1,prbs[i]);
        }
        printf("\n\n\tIndex table\n\n\t");
        for(i=0;i<nhalf;i++){
        printf("%3d,",iii[i]);
        }
        printf("\n");
/*** Print on the file  *******************************/
        fprintf(fo,"\n\tPRBS signals with %d shift reg.\n\n",n);
        for(i=0;i<N;i++){
            fprintf(fo,"\t%4d = %d\n",i+1,prbs[i]);
        }
        fprintf(fo,"\n\n\tIndex table\n\n\t");
        for(i=0;i<nhalf;i++){
            if(!(i%20)) fprintf(fo,"\n");
            fprintf(fo,"%2d,",iii[i]);
        }
        fprintf(fo,"\n");
}
/*** Functions ***********************************************************/
int irbit(iseed)
unsigned long *iseed;

{
        unsigned long newbit;
        newbit = (*iseed & IB1) >>fb1 ¢ (*iseed & IB2) >>fb2;
        *iseed = (*iseed << 1) | newbit;
        return (int) newbit;
}
int irbit8(iseed)
unsigned long *iseed;
{
        unsigned long newbit;
    newbit = (*iseed & IB1) >>fb1 ¢ (*iseed & IB2) >>fb2
                ¢ (*iseed & IB3) >>fb3 ¢ (*iseed & IB4) >>fb4;
        *iseed = (*iseed << 1) | newbit;
        return (int) newbit;
}
/*****     PRBS.C     ********************************************/


/*****     MFBS.C     **********************************************

   1. Program name
      MFBS.C
```

```
   2. Object
       To generate the multi frequency binary signal

   3. Subroutines used
        'FFT.C' for fast Fourier transform

   4. Output
       'MFBS.SIG' will include binary signals and frequency spectrum

   5. Date
       January 1988

 ------------------------------------------------------------------------

     Main variable listing

       binary[i]       ; Binary signal
       absfur[i]       ; Fourier transformed signal at each frequency
       vfreq[I]        ; Value of frequency
       binold[I]       ; Previously generated signal
       isign[J]        ; Sign of signals
       fourie[I]       ; Complex signal before or after fourier
                       ; transform
       pindex          ; Performance index
       uindex          ; Updated performance index

       NFREQ           ; Number of frequencies
       N               ; Number of signals generated
       nhalf           ; N/2
       itry,jtry       ; Index of trial set of signals
       it              ; Number of iterations
       ITMIN           ; Minimum number of iterations                */


#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#define TRUE    1
#define N       512
#define N2      9
#define NFREQ   8
#define ITMIN   15

double per_indx(int vf[],int bin[]);

main()
{
    int i,j,k,nhalf,jtry,isum,it,isign[100],ne;
```

```
    double uindex,pindex;
    double absfur[NFREQ],power,to_power;
    int binary[N],binold[N];
    static itry = 10;
    FILE *fo;
    char filein[]="mfbs.sig";
    static int vfreq[] = {1,2,4,8,12,16,24,32};
    struct complex fourie[N];

/*
  1.  File open for output
*/
       if((fo = fopen(filein,"w")) ==NULL) {
           printf("\nCan't open file for output.\n");
           printf("Check problem and try again.\n");
           exit(0);
       }
/*
  2.  Generate a half of initial binary signals by random number
*/
       nhalf = N/2;
       randomize();
       for(i=0;i<nhalf;i++) {
           if(random(2))
               binary[i] = 1;
           else
               binary[i] = -1;
       }
/*
  3.  Generate 2nd half of signals from 1st half
*/
       binary[nhalf] = binary[nhalf-1];
       for(i=nhalf+1;i<N;i++) {
           j = N-i+1;
           binary[i] = binary[j];
       }
/*
  4.  Repeat calculations until the best signals are found,
      but at least ITMIN times of iteration
*/
       it = 0;
       while(TRUE) {
           printf("---- ITERATION NO. = %2d\n",it);
           randomize();
           itry = random(nhalf);
           printf("\n\titry = %3d\n",itry);
           pindex = per_indx(vfreq,binary);
           uindex = pindex;
           jtry = itry;
/*
```

```
    4.1  Try for every signals
*/
            for(j=0;j<nhalf;j++) {
                binary[jtry] *= -1;
                ne = N -jtry + 1;
                if(jtry >1) {
                    if(jtry == (nhalf-1))
                        binary[jtry+1] = binary[jtry];
                    else
                        binary[ne] = binary[jtry];
                }
                pindex = per_indx(vfreq,binary);
/*
    4.2  Compare current value of performance index with previous one.
         If improved, change to new signal, otherwise restore the
         previouse signal.
*/
                if(pindex < uindex) {
                    uindex = pindex;
                    printf("\t Jtry = %3d, Updated P.I. = %10.5e\n",
                                jtry,uindex);
                }
                else {
                    binary[jtry] *= -1;
                    ne = N -jtry + 1;
                    if(jtry >1) {
                        if(jtry == (nhalf-1))
                            binary[jtry+1] = binary[jtry];
                        else
                            binary[ne] = binary[jtry];
                    }
                }
/*
        4.3  Try for next signal
*/
                jtry++;
                if(jtry >=nhalf)
                    jtry -= nhalf;
            }
/*
        4.5  If signals are not repeated or iteration time is less
             than ITMIN, repeat the iterations.
*/
            isum = 0;
            for(j=0;j<N;j++)
                isum += abs(binary[j] - binold[j]);
            printf("\n\tisum = %d\n",isum);
            if(isum == 0 && it >= ITMIN)
                break;
            else {
```

```
                for(j=0;j<N;j++)
                    binold[j] = binary[j];
                it++;
            }
        }
/*
  5.0  Prepare output
    5.1  Performance index and number of iteration times
*/
        pindex = per_indx(vfreq,binary);
        fprintf(fo,"Performance index = %10.6f\n",pindex);
        fprintf(fo,"Number of iteration = %4d\n",it);
/*
    5.2  Relative power at each frequency and
         Total power distributed at those frequencies.
*/
        fprintf(fo,"Freq./Sampling freq. Relative magnitude\n");
        for(i=0; i<N; i++) {
            fourie[i].x = binary[i] * 1.0;
            fourie[i].y = 0.;
        }
        fft(fourie,N2,-1);

        to_power = 0.;
        for(i=0; i<NFREQ; i++) {
            j = vfreq[i];
            absfur[i] = cabs(fourie[j])/N;
            to_power += 2.* absfur[i] * absfur[i];
            fprintf(fo,"\t%3d  %12.6f\n",
            vfreq[i],absfur[i]/absfur[0]);
        }
        fprintf(fo,"\tTotal power = %10.6f\n",to_power);
/*
    5.3  Final binary signals in simplified form
*/
        for(i=0;i<100;i++)
            isign[i] = 0;
        j=0;
        for(i=0;i<N;i++) {
            isign[j]++;
            if(i < N-1){
                if(binary[i+1] != binary[i])
                    j++;
            }
        }
        fprintf(fo,"\n\tMultifrequency Binary Signals\n");
        for(i=0;i<=j;i++) {
            if(i%2 ==0)
                fprintf(fo,"%3d(+) ",isign[i]);
            else
```

```
                fprintf(fo,"%3d(-) ",isign[i]);
            if((i%5)==4)
                fprintf(fo,"\n");
        }
        fprintf(fo,"\n\t\Multifrequency Binary Signals\n");
        for(i=0;i<N;i++) {
            fprintf(fo,"%2d,",binary[i]);
            if((i%20)==19)
                fprintf(fo,"\n");
        }
        fclose(fo);
}

double per_indx(vf,bin)
int vf[],bin[];
{
    int i,j;
    struct complex fourie[N];
    double power,pi,abs;

        for(i=0; i<N; i++) {
            fourie[i].x = bin[i] * 1.0;
            fourie[i].y = 0.;
        }
        fft(fourie,N2,-1);
        pi = 0.;
        for(i=0; i<NFREQ; i++) {
            j = vf[i];
            abs = cabs(fourie[j])/N;
            power = 2.* abs * abs;
            pi += (1./NFREQ - power) * (1./NFREQ - power);
        }
        return(pi);
}
/*****   MFBS.C   ************************************************\

/*****   FFT.C    ************************************************

    1. Program name
       FFT.C

    2. Date
       December 1987

    3. Object
       Fast Fourier transform or inverse transform

    4. Usage
       As a function of other program
```

```
    5. Arguments
       'data' for complex input and output
       'n2' for 2th power of total number of data
       'sign' for direction of transform
       ( sign = -1 for forward transform
         sign = 1 for backward transform)

-------------------------------------------------------------------/
#include <math.h>
#include <alloc.h>

#define     PI      3.141592653589795
#define     SWAP(a,b) temp=(a);(a)=(b);(b)=temp

void fft(data,n2,sign)
struct complex data[];
int n2,sign;
{
    int *m,nblock,iblock,lblock,lbhalf;
    int i,ii,j,jh,k,l,istart,n;
    struct complex wk,q;
    double v,temp;

        n = floor(pow(2,n2));
        m = (int *) malloc(sizeof (int));
        for(i=0;i<n2;i++)
            *(m+i)=floor(pow(2,(n2-i-1)));

        for(l=0;l<n2;l++){
            nblock=floor(pow(2,l));
            lblock=n/nblock;
            lbhalf=lblock/2;
            k=0;
            /*  Compute Fast fourier transform     */
            for(iblock=0;iblock<nblock;iblock++){
                v=sign*2.*PI*k/n;
                wk.x=cos(v);
                wk.y=sin(v);
                istart=lblock * iblock ;
                for(i=0;i<lbhalf;i++){
                    j = i + istart ;
                    jh= j + lbhalf ;
                    q.x=data[jh].x * wk.x - data[jh].y * wk.y;
                    q.y=data[jh].x * wk.y + data[jh].y * wk.x;
                    data[jh].x = data[j].x - q.x;
                    data[jh].y = data[j].y - q.y;
                    data[j].x = data[j].x + q.x;
                    data[j].y = data[j].y + q.y;
                }
                for(i=1;i<n2;i++){
```

```
                ii=i;
                if(k >= m[i])
                    k=k-m[i];
                else
                    break;
            }
            k=k+m[ii];
        }
    }
    /*  Reorder the output  */
    k=0;
    for(j=0;j<n;j++){
        if(k>j){
            SWAP(data[j].x,data[k].x);
            SWAP(data[j].y,data[k].y);
        }
        for(i=0;i<n2;i++){
            ii=i;
            if(k>=m[i])
                k=k-m[i];
            else
                break;
        }
        k=k+m[ii];
    }
    /*  Scaling back if back transform */
    if(sign >= 0){
        for(i=0;i<n;i++){
            data[i].x=data[i].x/n;
            data[i].y=data[i].y/n;
        }
    }
}
/*****    FFT.C    ************************************************/
```

APPENDIX E: PROGRAM LISTING FOR SIMULATION STUDY

```
/*****     SYSOUT.C     ******************************************

    1. Program name
       SYSOUT.C

    2. Date
       March 1988

    3. Object
       To simulate system using Runge-Kutta 4th classic
       differential equation solver

-------------------------------------------------------------------/
#include <math.h>
#include <alloc.h>
#include <stdio.h>

#define         NSMPLS  512

float process(int in);
int norder,lag,stptim;
float divi,*yold,*a,dt;

main()
{
    int i,ii,k,sign=1;
    float *tc,gain,deadtm,smpltm;
    float sysout[NSMPLS];
    int signal[NSMPLS],in;
    char file[13];
    FILE *fo;
    static int prbs[65] =  {
        28,16,12, 4,16, 8, 4, 4, 8, 8, 4, 8, 4,24, 4,12, 4, 8, 8,12,
        4, 4,12, 4, 4, 4, 8, 4, 8,20, 8, 8, 8, 4, 4, 4, 4, 8,12, 8,
        16, 4, 8, 4, 4,16, 4, 4, 4, 4, 4, 4,20, 4, 4, 8, 4, 4, 4,12,
        8, 4,12,16,NULL };
        ii=0;
        k=0;
/*      norder=3; */
        smpltm = 0.07;
        stptim = 10;
        printf("Enter the number of orders in transfer function : ");
        scanf("%d",&norder);
        printf("Enter the time constants  (float) : ");
        tc = (float *) calloc (norder,sizeof(float));
        a = (float *) calloc (norder,sizeof(float));
        yold = (float *) calloc (norder,sizeof(float));
```

```
        for(i=0;i<norder;i++)
            scanf("%f",tc+i);
/*      printf("Enter the gain and dead time (float) : ");
        scanf("%f %f",&gain,&deadtm); */
        gain = 1.0;
        deadtm= 0.0;
        divi = 1./gain;
        *(a+1) = 0.;
        for(i=0;i<norder;i++) {
            divi *= *(tc+i);
            *(a+1) += *(tc+i);
        }
        if(norder == 3)
            *(a+2) = (*tc+(*(tc+1))*(*(tc+2))+(*tc)*(*(tc+1)))/gain;
        else if(norder == 4) {
            *(a+2) = ((*tc+*(tc+1))*(*(tc+2)+*(tc+3))+(*tc)*(*(tc+1))
                    +(*(tc+2)*(*(tc+3))))/gain;
            *(a+3) = ((*tc+*(tc+1))*(*(tc+2))*(*(tc+3))+(*(tc+2)+
                    *(tc+3))*(*tc)*(*(tc+1)))/gain;
        }
        *a = 1./gain;
        *(a+1) /= gain;

/*      printf("Enter the sampling time in sec. (float) : ");
        scanf("%f",&smpltm);
        printf("Enter the number of steps per unit time (int) : ");
        scanf("%d",&stptim); */
        dt =  smpltm/stptim;

        for(i=0;i<NSMPLS;i++){
            if(k<prbs[ii]){
                signal[i]=sign;
                k++;
            }
            else{
                k=0;
                ii++;
                sign  *= -1;
                i--;
            }
        }
/*      for(i=0;i<NSMPLS;i++) {
                if(!(i%20)) printf("\n");
                printf("%2d,",signal[i]);
        } */

/*      printf("%f %f %f %f\n",smpltm,gain,*tc,*(tc+1)); */
        for(i=0;i<norder;i++)
            *(a+i) /= divi;
            *(yold+i) = 0.;
```

```
        for(i=0;i<norder;i++)
            printf("a = %f ",*(a+i));
        printf("\n");
        lag = deadtm/smpltm;
/*      printf("lag = %d\n",lag); */
/*      for(i=0;i<norder;i++)
        printf("a = %f\n",*(a+i)); */
        printf("Calculation is being performed ......\n");
        for(i=0;i<2*NSMPLS;i++) {
            if(i < lag) lag -= NSMPLS;
            if(i < NSMPLS) {
                in = signal[i-lag];
                process(in);
/*              printf("i = %d in = %2d out = %f\n",
                i,signal[i],process(in));  */
            }
            else {
                in = signal[i-NSMPLS-lag];
                sysout[i-NSMPLS] = process(in);
            }
        }
        printf("Do you want to print the results on the file (y/n)?");
        if ((getch()) == 'y') {
            printf("\nEnter the file name : ");
            scanf("%s",file);
            if((fo = fopen(file,"w")) !=NULL) {
                fprintf(fo,
                    "\t\tSequence\t Input\t Output\n");
                for(i=NSMPLS;i<2*NSMPLS;i++) {
                    if(i<lag) lag -= NSMPLS;
                    fprintf(fo,"\t%3d\t%2d\t%10.6f\n",
                    i-NSMPLS,signal[i-NSMPLS-lag],sysout[i-NSMPLS]);
                }
            }
        }
}

float process(int in)
{
    int i,ii,j,k;
    float cc,*ynew;
    float rk[4][5],sum,x;
    static float b[4] = {0.,0.5,0.5,1.};
/*      printf("norder = %d divi = %f dt = %f in = %2d stptim = %d\n",
            norder,divi,dt,in,stptim); */
        ynew = (float *) calloc(norder,sizeof(float));
        x=0.;
        for(ii=0;ii<stptim;ii++) {
            x += dt;
```

```
            for(i=0;i<norder;i++)
                rk[i][0] = 0.;
            for(i=1;i<5;i++) {
                sum = 0.;
                for(j=0;j<norder;j++)
                    sum += *(a+j)*(*(yold+j) + b[i-1]*rk[j][i-1]);
                    rk[norder-1][i] = dt * ( in/divi - sum);
                for(j=norder-2;j>=0;j--)
                    rk[j][i] = dt * (*(yold+j+1)+b[i-1]*rk[j+1][i-1]);
            }
            for(j=0;j<norder;j++)
               *(ynew+j) = *(yold+j)+
                    (rk[j][1]+2.*rk[j][2]+2.*rk[j][3]+rk[j][4])/6.;
            for(i=0;i<norder;i++)
                *(yold+i) = *(ynew+i);
        }
        return(*ynew);
}
/*****     SYSOUT.C     *******************************************/


/*****     NOISE.C     ********************************************
```

1. Program name
   NOISE.C

2. Date
   January 1988
   March 29 1988, revised

3. Object
   Generation of low pass filtered random noise

4. Subroutines used
   'FFT' for fast fourier transform

5. Input
   Number of filter order

6. Output to 'NOISE.DAT' file
   a. Absolute mean of noise
   b. Noise

I. Main program

1) Main variables listing

```
ran_nos[]   ; Randomly selected noise
hf[]        ; Low pass filter coefficients
```

```
        flt_nos[]   ; Filtered noise
        temp[]      ; Before or after transformed random complex noise
        be_flt[]    ; Before or after transformed nonfiltered complex
                      noise
        af_fit[]    ; Before or after transformed filtered complex
                      noise
        NSMPLS      ; Number of samples
        N2          ; Power of 2, log 2 of Nsmpls
        CF          ; Cut off frequency divided by natural frequency
        SMPLTM      ; Sampling interval
        no          ; Number of filter length

    2) Program starts                                                  */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NSMPLS  512
#define N2      9
#define CF      16.
#define SMPLTM  1.
#define PI      3.141592654

void lofltr(int no,double hf[]);

main()
{
    int i,no;
    float sum,ran_nos[NSMPLS],flt_nos[NSMPLS],w,avg,abs_sum,abs_avg;
        float avg_off,be_amp,af_amp;
        double hf[NSMPLS];
    struct complex temp[NSMPLS],be_flt[NSMPLS],af_flt[NSMPLS];
    FILE *fo;
    char filein[] = "noise.dat";

        fo = fopen(filein,"w");
        printf("Enter the number of filter order, no = ");
        scanf("%d",&no);
/*
    3) Select the random noise by random number generator
       and try to make the mean value zero
*/
        sum=0.;
        randomize();
        for(i=0;i<NSMPLS*2;i++) {
            ran_nos[i] = rand()/32767. - 0.5;
            sum += ran_nos[i];
        }
```

```
        avg_off = sum/(NSMPLS*2);
        for(i=0;i<NSMPLS;i++) {
            ran_nos[i] -= avg_off;
            temp[i].x = ran_nos[i];
            temp[i].y = 0.;
        }
/*
    4) Transform the data by FFT
*/
        fft(temp,N2,-1);
/*
    5) Calculate the multiplying coefs. by transfer function
       of low pass filter and multiply them to the transformed
       data
*/
        lofltr(no,hf);
        for(i=0;i<NSMPLS;i++) {
            temp[i].x *= hf[i];
            temp[i].y *= hf[i];
        }
/*
    6) Transform the products back
*/
        fft(temp,N2,1);
        for(i=0;i<NSMPLS;i++)
            flt_nos[i] = temp[i].x;
/*
     7) Transform the raw and the filtered noise for comparison
        and record those
*/
        for(i=0;i<NSMPLS;i++) {
            be_flt[i].x = ran_nos[i];
            af_flt[i].x = flt_nos[i];
            be_flt[i].y = 0.;
            af_flt[i].y = 0.;
        }
        fft(be_flt,N2,-1);
        fft(af_flt,N2,-1);
        for(i=0;i<NSMPLS;i++) {
            w = 2*PI*(i+1)/(NSMPLS*SMPLTM);
            be_amp = cabs(be_flt[i]);
            af_amp = cabs(af_flt[i]);
            printf("%10.6f %12.6f %12.6f\n",w,be_amp,af_amp);
        }
/*
     8) Calculate the mean and the absolute mean value for reference
*/
        abs_sum = 0.;
        sum=0.;
        for(i=0;i<NSMPLS;i++) {
```

```
            abs_sum += fabs(flt_nos[i]);
            sum += flt_nos[i];
        }
        avg = sum/NSMPLS;
        abs_avg = abs_sum/NSMPLS;
/*
     9) Record output
*/
        fprintf(fo,"\tSum of signal = %8.4f\n",sum);
        fprintf(fo,"\tAbsolute sum of signal = %8.4f\n",abs_sum);
        fprintf(fo,"\tMean of signal = %8.4f\n",avg);
        fprintf(fo,"\tAbsolute mean of signal = %8.4f\n",abs_avg);
        fprintf(fo,"\n\t\tNoise data\n\n\tFilterorder = %3d\n",no);
        for(i=0;i<NSMPLS;i++) {
            fprintf(fo,"%10.6f  ",flt_nos[i]);
            if((i%6) == 5)
                fprintf(fo,"\n");
        }
        fclose(fo);
}
/*-------------------------------------------------------------------
    Subroutine for low pass filter with window                      */

void lofltr(norder,flt_coef)
int norder;
double flt_coef[];
{
    int i,j;
    float fs,delf,f;
        fs = CF/NSMPLS;
        delf = 1./NSMPLS;
        f = 0.;
        for(i=0;i<NSMPLS;i++) {
            flt_coef[i] = 2.*fs;
            for(j=1;j<=norder;j++)
                flt_coef[i] += sin(PI*j/norder)/(PI*j/norder)*
                               2./(PI*j)*sin(2.*PI*j*fs)*
                               cos(2.*PI*j*f);
            f += delf;
        }
}
/*****     NOISE.C     ***********************************************/
```

APPENDIX F: PROGRAM LISTING FOR MODEL BUILDING PACKAGE

```
/*****     MENU.C     ************************************************

    1. Program name
       MENU.C

    2. Date
       June 1988
       Aug  1988  Rev. 1
       Sep  1988  Rev. 2

    3. Object

       To make a handy software package for the
       following objetcs using pop-up window

          1) To send input signals (step, PRBS, and MFBS)
             to the diesel engine fuel metering
             through the Micro-mac interface.
          2) To measure the smoke output from the diesel
             engine through the Micro-mac interface.
          3) To measure the temperatures at various
             lacations in diesel engine
          4) To show graph for collected data
          5) To store above results on floppy disk except
             graphics.

     4. External Function used
        An assembly routine "TMCM.ASM"

     5. Child program used
        1) "SIGNXXX.C" for a title and a short help
        2) "LOG.C" for data collection
        3) "CGRAPHIO.C" for graphics

     6. Files name convention
        1) 'log*****.dat' will include the input/outpue
           data from diesel engine.
        2) Speed : 4th characters in file name
           l : 1000 rpm
           m : 1500 rpm
           h : 2000 rpm
        3) Torque : 5th characters in file name
           l : 30% load
           m : 60% load
           h : 90%l load
        4) Replication : 8th character in file name
           1, 2 and 3
```

```
        5) Input signal
           s : Step input
           p : PRBS
           m : MFBS
        Ex.) logllls.dat : File from data logger at 1000
             rpm, 30% load for replication 1 with step input

-------------------------------------------------------------------*/

/* include files */
#include <bios.h>
#include <dos.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "myfile.h"
#include <math.h>
#include <conio.h>
#include <stdarg.h>
#include <graphics.h>

/* Ascii key definitions */
#define ENTER       13
#define SPACE       32
#define ESC         27

/* Scan key redefinitions */
#define UPPER_ARROW   72+'z'
#define LOWER_ARROW   80+'z'

/* limitations: */
#define     MENUNUMBER       11     /* number of menu            */
#define     MAXMENUITEMS     5      /* number of menu items      */
/* macros to make the code more readable                         */
#define     ring_bell()         putch(7) /* bell-ringing macro  */
#define     VIDEO_INT           0x10     /* Bios video interrupt*/
#define     SET_CURSOR_SIZE     0x01     /* Int 10 function 1   */
#define     GET_CURSOR_POS      0x02     /* Int 10 function 2   */
#define     READ_CURSOR         0x03     /* Int 10 function 3   */
#define     CURSOR_SAVE         0        /* macro for cursor save*/
#define     CURSOR_RESTORE      1        /* macro for cursor store*/

#define     SCREEN_SIZE         2000     /* screen size 2000 words */
#define     VIDEO_INT           0x10     /* Bios video interrupt   */
#define     GET_VID_MODE        0x0f     /* Int 10 function F      */
#define     MONO_MODE           7        /* monochrome video mode  */
#define     MONO_SCREEN         0xb0000000/* address of mono screen */
#define     COLOR_SCREEN        0xb8000000/* address of color screen*/
```

```
void inicom();
void setsnd();
void clesnd();
char *chasnd();
void degsnd();
char *scasnd();

void exitf(int run_i);
void display_make();
void display_cle();
int Run(int i,int itemnumber);
void border_single(int startx,int starty, int endx, int endy);
void border_double(int startx,int starty, int endx, int endy);
void cursor(int op);
void initialize(int run_i,int *choices,char *initials);
int key_select(int run_i,int mark);

/* global objects: */
char *torque,*speed,*rep,*input;
int choices;
char initials[MAXMENUITEMS];
int far *screen;                        /* pointer to screen buffer */
int old_screen[SCREEN_SIZE];            /* store old screen here    */

static char *file_appn = ".dat";
static char *file_head[3] = {"log","frq","cur"};
char *file_part[4],file_name[3][12];

/* list of function declarations for cue_function */
int    main_stf(),main_cof(),main_anf(),main_grf();
int    anal_trf(),anal_cff();
int    curve_2df(),curve_3df(),curve_40f();
int    set_idf(),set_tsf(),set_temf();
int    iden_spf(),iden_tqf(),iden_rpf(),iden_inf();
int    test_fif(),test_smf(),test_bof();
int    speed_10f(),speed_15f(),speed_20f();
int    torque_30f(),torque_60f(),torque_90f();
int    rep_1f(),rep_2f(),rep_3f();
int    stepf(),prbsf(),mfbsf();
int    tm_dmf(),frq_dmf(),pos_cur();

/* array of structures with names, descriptions and function pointers*/
struct _cue
{
    char    *cue_name;                  /* name to be displayed   */
    int     (*cue_function)();          /* function to be called  */
}
cue[MENUNUMBER][MAXMENUITEMS] = {
```

```
{/* 0. Main menu */
/* --12345678901234567890l----12345678901234567890l2----for alignment*/
    "       Set Up         ",  main_stf,
    "   Data Collection    ",  main_cof,
    "    Data Analysis     ",  main_anf,
    "      Graphics        ", main_grf,
    NULL,NULL
},
{/* 1. Analysis menu */
    "   Freq. Response     ",anal_trf,
    "   Curve Fitting      ",anal_cff,
    NULL,NULL
},
{/* 2. Curve fitting menu */
    "     2nd w/ Dead      ", curve_2df,
    "     3rd w/ Dead      ", curve_3df,
    "     4th w/o Dead     ", curve_40f,
    NULL,NULL
},
{/* 3. Setup menu */
    "  Identification    ", set_idf,
    "    Test I/O        ", set_tsf,
    " Check Temperature ", set_temf,
    NULL,NULL
},
{/* 4. Identification menu */
    "     Speed          ", iden_spf,
    "     Torque         ", iden_tqf,
    "    Replication     ", iden_rpf,
    "   Input signal     ", iden_inf,
    NULL,NULL
},
{/* 5. Test I/O menu */
    "  Fuel Injection   ", test_fif,
    "     Smoke Out     ", test_smf,
    "       Both        ", test_bof,
    NULL,NULL
},
{/* 6. Speed menu */
    "  1000 rpm  ", speed_10f,
    "  1500 rpm  ", speed_15f,
    "  2000 rpm  ", speed_20f,
    NULL,NULL
},
{/* 7. Torque menu */
    "    30% load     ", torque_30f,
    "    60% load     ", torque_60f,
    "    90% load     ", torque_90f,
    NULL,NULL
},
```

```
{/* 8. Replication menu */
    "   1   ", rep_1f,
    "   2   ", rep_2f,
    "   3   ", rep_3f,
    NULL,NULL
},
{/* 9. Inpurt signal menu */
    "    Step input     ", stepf,
    "   Pseudo Random   ", prbsf,
    "  Multi Frequency  ", mfbsf,
    NULL,NULL
},
{/* 10. Graph menu */
    "     Time domain     ", tm_dmf,
    "   Frequency domain  ", frq_dmf,
    "   Postualted Curve  ", pos_cur,
    NULL,NULL
}
};

/*--------------------------------------------------------------------*/
/* functions */
main_stf()
{
        me_nu(22,2,42,7,3);
}
main_cof()
{
    char *pathname = "B:\\LOG.EXE";
    char *args[] = {"LOG.EXE",file_name[0],NULL};

    display_make();
    gotoxy(22,1);
    textattr(BLUE*16|WHITE);
    cputs(" DATA COLLECTION FOR THE MODELLING ");
    spawnv(P_WAIT,pathname,args);
    display_cle();
}

main_anf()
{
    me_nu(20,6,42,10,1);
}

main_grf()
{
    me_nu(20,7,42,12,10);
}

anal_trf()
```

```
{
    char *pathname = "B:\\FRQ.EXE";
    char *args[] = {"FRQ.EXE",file_name[0],file_name[1],NULL};

    display_make();
    gotoxy(12,1);
    textattr(BLUE*16|WHITE);
    cputs(" CALCULATION OF FREQUENCY RESPONSE FOR TRANSFER FUNCTION ");
    spawnv(P_WAIT,pathname,args);
    display_cle();
}

anal_cff()
{
    me_nu(38,8,60,13,2);
}

curve_2df()
{
    char *pathname = "B:\\CUR.EXE";
    char *args[] = {"cur.exe",file_name[1],file_name[2],
                                        "1","2",NULL};

    display_make();
    gotoxy(16,1);
    textattr(BLUE*16|WHITE);
    cputs(" MODEL PARAMETER SEARCHING (2ND WITH DEAD-TIME) ");
    spawnv(P_WAIT,pathname,args);
    display_cle();
}
curve_3df()
{

    char *pathname = "B:\\CUR.EXE";
    char *args[] = {"cur.exe",file_name[1],file_name[2],
                    "1","3",NULL};

    display_make();
    gotoxy(16,1);
    textattr(BLUE*16|WHITE);
    cputs(" MODEL PARAMETER SEARCHING (3RD WITH DEAD-TIME) ");
    spawnv(P_WAIT,pathname,args);
    display_cle();
}

curve_40f()
{

    char *pathname = "B:\\CUR.EXE";
    char *args[] = {"cur.exe",file_name[1],file_name[2],
```

```
                                        "0","4",NULL};

    display_make();
    gotoxy(16,1);
    textattr(BLUE*16|WHITE);
    cputs(" MODEL PARAMETER SEARCHING (4th W/O DEAD-TIME) ");
    spawnv(P_WAIT,pathname,args);
    display_cle();
}

set_idf()
{
    me_nu(40,1,59,7,4);
}

set_tsf()
{
    me_nu(40,4,59,9,5);
}

set_temf()
{
    int i;

    window(42,6,64,13);
    textbackground(LIGHTGRAY);
    clrscr();
    textattr(LIGHTGRAY*16|RED);
    gotoxy(2,2);
    cputs("Temperature (deg. C)");
    textattr(LIGHTGRAY*16|BLACK);
    gotoxy(2,4);
    cputs("Engine Oil : ");
    gotoxy(2,5);
    cputs("Coolant    : ");
    gotoxy(2,6);
    cputs("Inlet Air  : ");
    gotoxy(2,7);
    cputs("Exh. Gas   : ");
    while(!kbhit())
    {
        gotoxy(14,4);
        cprintf(" %7s ",strtok(scasnd(),"/"));
        for(i=1;i<4;i++){
            gotoxy(14,4+i);
            cprintf(" %7s",strtok(NULL,"/"));
        }
    }
    if(getch() == ESC)
        exitf(5);
```

```
}
iden_spf()
{
    me_nu(58,2,71,7,6);
}

iden_tqf()
{
    me_nu(58,3,75,8,7);
}

iden_rpf()
{
    me_nu(58,4,66,9,8);
}

iden_inf()
{
    me_nu(58,5,78,10,9);
}

speed_10f()
{
    speed = cue[6][0].cue_name;
    file_part[0] = "l";
}

speed_15f()
{
    speed = cue[6][1].cue_name;
    file_part[0] = "m";
}

speed_20f()
{
    speed = cue[6][2].cue_name;
    file_part[0] = "h";
}


torque_30f()
{
    torque = cue[7][0].cue_name;
    file_part[1] = "l";
}

torque_60f()
{
    torque = cue[7][1].cue_name;
    file_part[1] = "m";
```

```
}

torque_90f()
{
    torque = cue[7][2].cue_name;
    file_part[1] = "h";
}

rep_1f()
{
    rep = cue[8][0].cue_name;
    file_part[2] = "1";
}

rep_2f()
{
    rep = cue[8][1].cue_name;
    file_part[2] = "2";
}

rep_3f()
{
    rep = cue[8][2].cue_name;
    file_part[2] = "3";
}
stepf()
{
    input = cue[9][0].cue_name;
    file_part[3] = "s";
}

prbsf()
{
    input = cue[9][1].cue_name;
    file_part[3] = "p";
}

mfbsf()
{
    input = cue[9][2].cue_name;
    file_part[3] = "m";
}

test_fif()
{
    int key;

    window(59,8,80,12);
    textbackground(LIGHTGRAY);
    clrscr();
```

```
    textattr(LIGHTGRAY*16|RED);
    gotoxy(6,2);
    cputs(" Signal Test");
    textattr(LIGHTGRAY*16|BLACK);
    while(TRUE)
    {
        if(!(key = tolower(getch())))  /* get key, and lowercase   */
        key = (getch() + 'z');         /* Not Ascii? get scan code */
        switch(key)                    /* which key was pressed?   */
        {
            case ESC:                  /* if ESC                   */
                exitf(11);
                break;
            case LOWER_ARROW:          /* if LOWER_ARROW           */
                setsnd();
                gotoxy(4,4);
                cputs("Input : set (lower)");
                break;
            case UPPER_ARROW:          /* if UPPER_ARROW           */
                clesnd();
                gotoxy(4,4);
                cputs("Input : cle (upper)");
                break;
            default:
                ring_bell();
                break;
        }
    }
}

test_smf()
{
    int key;

    window(59,8,80,12);
    textbackground(LIGHTGRAY);
    clrscr();
    textattr(LIGHTGRAY*16|RED);
    gotoxy(6,2);
    cputs(" Smoke Test");
    textattr(LIGHTGRAY*16|BLACK);
    gotoxy(4,4);
    cputs("Smoke : ");
    while(!kbhit())
    {
        gotoxy(11,4);
        cprintf(" %5.2f (%) ",100*atof(chasnd()));
    }
    if(getch() == ESC)
        exitf(11);
```

```
}

test_bof()
{
    int key;
    window(59,8,80,12);
    textbackground(LIGHTGRAY);
    clrscr();
    textattr(LIGHTGRAY*16|RED);
    gotoxy(7,2);
    cputs(" I/O Test");
    textattr(LIGHTGRAY*16|BLACK);
    gotoxy(4,4);
    cputs("Smoke : ");
    while(TRUE)
    {
        if(!(key = tolower(getch())))  /* get key, and lowercase   */
        key = (getch() + 'z');         /* Not Ascii? get scan code */
        switch(key)                    /* which key was pressed?   */
        {
            case ESC:                  /* if ESC                   */
                exitf(11);
                break;
            case LOWER_ARROW:          /* if LOWER_ARROW           */
                setsnd();
                gotoxy(4,3);
                cputs("Input : set (lower)");
                while(!kbhit())
                {
                    gotoxy(11,4);
                    cprintf(" %5.2f (%)",100*atof(chasnd()));
                }
                break;
            case UPPER_ARROW:          /* if UPPER_ARROW            */
                clesnd();
                gotoxy(4,3);
                cputs("Input : cle (upper)");
                while(!kbhit())
                {
                    gotoxy(11,4);
                    cprintf(" %5.2f (%)",100*atof(chasnd()));
                }
                break;
            default:
                ring_bell();
                break;
        }
    }
}
```

```
tm_dmf()
{
    char *pathway= "B:\\CGRAPHIO.EXE";
    char *argrp[] = {"CGRAPHIO.EXE",file_name[0],NULL};

    screen_sav();
    spawnv(P_WAIT,pathway,argrp);
    return_to_menu();
}

frq_dmf()
{
    char *pathway= "B:\\CGRAPH.EXE";
    char *argrp[] = {"CGRAPH.EXE",file_name[1],NULL};

    screen_sav();
    spawnv(P_WAIT,pathway,argrp);
    return_to_menu();
}

pos_cur()
{
    char *pathway= "B:\\CGRAPH.EXE";
    char *argrp[] = {"CGRAPH.EXE",file_name[1],file_name[2],NULL};

    screen_sav();
    spawnv(P_WAIT,pathway,argrp);
    return_to_menu();
}

void display_make()
{
    textattr(LIGHTGRAY*16|BLACK);
    window(1,13,80,25);
    border_single(1,13,80,25);
    gotoxy(36,1);
    cputs(" Display ");
    gotoxy(30,12);
    cputs(" Press a key for the menu ");
    window(2,14,79,23);
    textbackground(LIGHTGRAY);
    clrscr();
}
void display_cle()
{
    getch();
    window(1,13,80,25);
    textbackground(BLACK);
    clrscr();
}
```

```
void exitf(int run_i)
{
    int i,j;

    textbackground(BLACK);
    switch(run_i)
    {
        case 0:
            window(1,1,80,25);
            clrscr();
            cursor(CURSOR_RESTORE);
            exit(0);
        case 1:
        case 3:
            window(1,1,80,25);
            clrscr();
            me_nu(2,3,24,9,0);
        case 2:
            window(38,8,80,25);
            clrscr();
            me_nu(20,6,42,10,1);
        case 4:
            window(40,1,80,25);
            clrscr();
            window(43,2,72,8);
            clrscr();
            textattr(LIGHTGRAY*16|RED);
            border_single(43,2,72,8);
            gotoxy(12,1);
            cputs(" Verify ");
            window(44,3,71,7);
            textattr(LIGHTGRAY*16|YELLOW);
            gotoxy(1,1);
            cprintf(" Speed  :   %12s    " ,speed);
            gotoxy(1,2);
            cprintf(" Torque : %12s     ",torque);
            gotoxy(1,3);
            cprintf(" Rep.   :%12s       ",rep);
            gotoxy(1,4);
            cprintf(" Signal :%12s",input);
            for(i=0;i<3;i++){
                strcpy(file_name[i],file_head[i]);
                for(j=0;j<4;j++)
                    strcat(file_name[i],file_part[j]);
                strcat(file_name[i],file_appn);
                strcat(file_name[i],NULL);
            }
            me_nu(22,2,42,7,3);
        case 5:
            window(40,1,80,25);
```

```
            clrscr();
            me_nu(22,2,42,7,3);
        case 10:
            window(20,7,42,11);
            clrscr();
            me_nu(2,3,24,9,0);
        case 11:
            window(58,8,80,12);
            clrscr();
            me_nu(40,4,59,9,5);
        default:
            window(40,1,80,25);
            clrscr();
            me_nu(40,1,59,7,4);
    }
}

/* to runs the appropriate function at cue[itemnumber].cue_function.*/
int Run(int i,int itemnumber)
{
    return(cue[i][itemnumber].cue_function)();
}
void border_single(int startx,int starty, int endx, int endy)
{
    register int i;

    textcolor(WHITE);
    gotoxy(1,1);
    putch('Z');
    for(i=1; i<endx-startx;i++)
    putch('D');
    putch('?');
    for(i=2; i<endy-starty; i++){
        gotoxy(1,i);
        putch('3');
        gotoxy(endx-startx+1, i);
        putch('3');
    }
    gotoxy(1,endy-starty);
    putch('@');
    for(i=1;i<endx-startx;i++)
        putch('D');
    putch('Y');
}
void border_double(int startx,int starty, int endx, int endy)
{
    register int i;

    textcolor(WHITE);
    gotoxy(1,1);
```

```
    putch('U');
    for(i=1; i<endx-startx;i++)
    putch('M');
    putch('8');
    for(i=2; i<endy-starty; i++){
        gotoxy(1,i);
        putch('3');
        gotoxy(endx-startx+1, i);
        putch('3');
    }
    gotoxy(1,endy-starty);
    putch('T');
    for(i=1;i<endx-startx;i++)
        putch('M');
    putch('>');
}
void cursor(int op)
{
    union REGS r;
    static int oldcurrow,oldcurcol,oldcurstart,oldcurend;

    if(op)                                  /* restore cursor        */
    {
        r.h.ah = GET_CURSOR_POS;            /* cursor position       */
        r.h.dh = oldcurrow;                 /* set the row           */
        r.h.dl = oldcurcol;                 /* set the column        */
        r.h.bh = 0;                         /* for video page 0      */
        int86(VIDEO_INT,&r,&r);             /* call the bios         */

        r.h.ah = SET_CURSOR_SIZE;           /* cursor size function  */
        r.h.ch = oldcurstart;               /* set the start         */
        r.h.cl = oldcurend;                 /* set the end           */
        r.h.bh = 0;                         /* for video page 0      */
        int86(VIDEO_INT,&r,&r);             /* call the bios         */
    }
    else                                    /* save the cursor       */
    {
        r.h.ah = READ_CURSOR;               /* cursor read function  */
        r.h.bh = 0;                         /* page 0                */
        int86(VIDEO_INT,&r,&r);             /* call the bios         */
        oldcurrow = r.h.dh;                 /* save the cursor row   */
        oldcurcol = r.h.dl = 0;             /* save the cursor column*/
        oldcurstart = r.h.ch;               /* save the cursor start */
        oldcurend = r.h.cl;                 /* save the cursor end   */
        r.h.ah = SET_CURSOR_SIZE;           /* cursor size function  */
        r.h.ch = 0x20;                      /* set bit 5 to hide     */
        int86(VIDEO_INT,&r,&r);             /* call the bios         */
    }
}
```

```
/* screen_sav()
 * saves the screen buffer in the old_screen array
 * and save the cursor */
screen_sav()
{
    int i;

    get_screen_mode();                          /* set screen pointer   */
                                                /* save old screen */
    for( i = 0; i < SCREEN_SIZE; i++)
        old_screen[i] = *(screen+i);
}

/* get_screen_mode()
 * returns the current screen mode in AL, and sets the screen pointer
 * to the appropriate video buffer address.
 */
get_screen_mode()
{
    union REGS r;

    r.h.ah = GET_VID_MODE;                      /* get video mode function*/
    int86(VIDEO_INT,&r,&r);                     /* call the bios          */
                                                /* set screen pointer     */
    screen=(int far *)((r.h.al==MONO_MODE)?MONO_SCREEN:COLOR_SCREEN);
}

return_to_menu()
{
    int i;
    union REGS r;

    for( i = 0; i < SCREEN_SIZE; i++)  /* restore original screen */
        *(screen+i) = old_screen[i];
    r.h.ah = SET_CURSOR_SIZE;                   /* cursor size function     */
    r.h.ch = 0x20;                              /* set bit 5 to hide cursor */
    int86(VIDEO_INT,&r,&r);                     /* call the bios            */
}

/* initialize()
 * initializes the initials array, sets number of choices found */
void initialize(int run_i,int *choices,char *initials)
{
    int i;

    for( i = 0; cue[run_i][i].cue_name && i < MAXMENUITEMS; i++)
        initials[i] = tolower(cue[run_i][i].cue_name[i+1]);
    initials[i] = NULL;
    *choices = i;
}
```

```
int key_select(int run_i,int mark)
{
    int key;
    char *match;

    if(!(key = tolower(getch())))      /* get key, and lowercase   */
    key = (getch() + 'z');          /* Not Ascii? get scan code */
    switch(key)                        /* which key was pressed?   */
    {
        case ENTER:                    /* if ENTER                 */
            Run(run_i,mark);           /* run function at mark     */
            break;
        case LOWER_ARROW:              /* if LOWER_ARROW/SPACE bar */
        case SPACE:
            mark = ((mark+1) % choices); /* move to next choice    */
            break;
        case UPPER_ARROW:               /* if UPPER_ARROW          */
                                        /* move to previous choice */
            mark = ((mark+choices-1) % choices);
            break;
        case ESC:                      /* if ESC                   */
            exitf(run_i);           /* goto previous screen*/
        default:                       /* if initials match        */
            if((match = strchr(initials,key)) != 0)
            {
                mark = (match-initials);
                if(mark==(choices-1))
                    return Run(run_i,mark);
                else
                    Run(run_i,match-initials);
             }
             else
                ring_bell();           /* bad choice, ring the bell*/
             break;
    }
    return(mark);
}

me_nu(int left,int top,int right,int bottom,int run_i)
{
    int mark;
    int  row, i;
    char *match;

    initialize(run_i,&choices,initials); /* initialize & save screen*/
    mark = 0;
    textbackground(BLUE);
    window(left,top,right,bottom);
    border_double(left,top,right,bottom);
```

```
switch(run_i)
{
    case(0):
        textbackground(RED);
        gotoxy(7,1);
        cputs(" Main Menu ");
        break;
    case(1):
        textbackground(RED);
        gotoxy(7,1);
        cputs(" Analysis ");
        break;
    case(2):
        textbackground(RED);
        gotoxy(7,1);
        cputs(" Eq. Order ");
        break;
    case(3):
        textbackground(RED);
        gotoxy(7,1);
        cputs(" Set-Up ");
        break;
    case(4):
        textbackground(RED);
        gotoxy(7,1);
        cputs(" Iden. ");
        break;
    case(5):
        textbackground(RED);
        gotoxy(6,1);
        cputs(" Pre-Test ");
        break;
    case(6):
        textbackground(RED);
        gotoxy(4,1);
        cputs(" Speed ");
        break;
    case(7):
        textbackground(RED);
        gotoxy(6,1);
        cputs(" Torque ");
        break;
    case(8):
        textbackground(RED);
        gotoxy(3,1);
        cputs(" Rep ");
        break;
    case(9):
        textbackground(RED);
```

```
            gotoxy(4,1);
            cputs(" Input signal ");
            break;
        case(10):
            textbackground(RED);
            gotoxy(6,1);
            cputs(" Graphics ");
            break;
        default:
            break;
    }
    while(TRUE)
    {
        window(left+1,top+1,right-1,bottom-1);
        for( i = 0; i < choices; i++){
            if(mark==i)
                textattr(LIGHTGRAY*16 | BLACK);
            else
                textattr(BLUE*16 |WHITE);
            gotoxy(1,i+1);
            cputs(cue[run_i][i].cue_name);
        }
        mark = key_select(run_i,mark);
    }
}
/*-----------------------------------------------------------------*/
main()                          /* main program                    */
{
    char *pathname = "B:\\SIGNCGA.EXE";
    char *args[] = {"SIGNCGA.EXE",NULL};

    spawnv(P_WAIT,pathname,args);
    window(1,1,80,25);
    clrscr();
    cursor(CURSOR_SAVE);
    inicom();
    me_nu(2,3,24,9,0);          /* called as needed                */
}
/*****     MENU.C     *********************************************/


/*****     LOG.C     ***********************************************

    1. Program name
       LOG.C

    2. Date
       May  2, 1988
       August 21, 1988 Revised
       September 1, 1988 Revised
```

```
        September 27, 1988 Revised

     3. Objective
        1) To send step, PRBS, and MFBS signals to
           the diesel engine fuel injection system
           through the the Micro-mac interface.
        2) To receive the smoke data from the diesel
           exhaust smoke meter through the Micro-mac
           interface.
        3) To receieve the temperature reading
        4) To store the data on the floppy disk for
           later analysis.

      4. Usage
         As a child program of "MENU.C" for collecting data.

      5. External Progam
         An assembly routine "tmcm.asm"

---------   Preprocessor  ----------------------------------------*/

#include <bios.h>
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#define     NUMSIG              512
#define     SETDAT               9

void tm_set(int n);
void con_sgn(int index[]);
void inicom();
void setsnd();
void clesnd();
char *chasnd();
char *scasnd();

int signal[NUMSIG];

/********     Main Program Starts ***********************************/
main(int argc, char *argv[])
{
/****** To declare and initialize the variables                     */
    FILE *fo;
    struct date today;
    struct time now;
    int n,latch;
```

```
    int i,j,k;
    float ii;
    double rawdata[SETDAT][NUMSIG];
    double data[NUMSIG];
    static char space[21] = "                    ";
    const int NEWLAT = 11931;           /* 100 pulses per 1 sec.  */
    const int OLDLAT = 65535;           /* 18.2 pulses per 1 sec. */
    int SMPTM ;               /* SMPTM*10 ms sampling time */
    double temp[2][4];

    /* Step input signal index */
    static int step[3] = {256,256,NULL};

    /* Pseudo random binary sequence index  7 bit*/
    /*static int prbs[130] =  {32, 8, 4,16, 4, 4, 4, 8,20, 4,
        4, 4, 4, 4, 4,12,20, 8,12,
        4, 4, 4, 4, 8, 8, 8, 8, 4, 4,24, 4,16, 8, 8, 4,12, 4,12, 8,
        4, 4, 4, 4, 4, 8, 4, 4, 4,12, 4, 8, 4, 4, 8, 4, 4,12, 8, 8,
        12, 8,16,12, 8, 4, 8,16, 4,12, 4, 4,12, 4, 4, 4,16, 4, 8, 4,
        20,16, 8, 4, 4, 8, 8, 4, 4, 4, 8, 4, 8, 4, 4, 4, 4,20, 4, 8,
        12, 4, 8, 8, 4, 8, 4, 8, 8,24,12, 4, 4, 8, 4,12,12,12, 4,28,
        4, 4, 8,12,16, 4, 4,16, 4,NULL}; */
    static int prbs[65] = {28,16,12, 4,16, 8, 4,
        4, 8, 8, 4, 8, 4,24, 4,12, 4, 8, 8,12,
        4, 4,12, 4, 4, 4, 8, 4, 8,20, 8, 8, 8, 4, 4, 4, 4, 8,12, 8,
        16, 4, 8, 4, 4,16, 4, 4, 4, 4, 4, 4,20, 4, 4, 8, 4, 4, 4,12,
        8, 4,12,16,NULL};


    /* Multifrequency binary sequence index, 512 signal */
    /* static int mfbs[28]={58,14,40,48,114,10,34,22,66,38,2,4,2,121,
        2,4,2,38,66,22,34,10,114,48,40,14,57,NULL}; */
    static int mfbs[32] = {31, 4,26,23,13, 7,39, 8,10,10,
                           28,21, 7,13, 4,25, 4,13, 7,21,
                           28,10,10, 8,39, 7,13,23,26, 4,
                           30,NULL};

    window(2,14,79,23);

/****** To convert index tabled signal to individual signal         */

    if(strpbrk(argv[1],"s"))
        {
            ccn_sgn(step);
            SMPTM = 20;
        }
    else if(strpbrk(argv[1],"p"))
        {
            con_sgn(prbs);
            SMPTM = 7;
```

```
        }
    else
        {
            con_sgn(mfbs);
            SMPTM = 7;
        }
    textattr(LIGHTGRAY*16|BLACK);

/****** To open a file and write a heading                          */
    if( (fo = fopen(argv[1],"w") ) != NULL)
        {
        fprintf(fo,
        "\n\n%s Data Table for System Identification\n\n",space);
        getdate(&today);
        fprintf(fo,
                  "%s%s%sDate: %d/%d/%d\n",space,space,space,
                  today.da_mon,today.da_day,today.da_year);
        gettime(&now);
        fprintf(fo,
           "%s%s%sTime: %d:%d:%d.%d\n",space,space,space,
             now.ti_hour,now.ti_min,now.ti_sec,now.ti_hund);
        }
    else
        cputs("Data file can not be open.");

    gotoxy(10,3);
    cprintf("Data file name is \"%s\"",argv[1]);


/****** To initialize the serial port                               */
    inicom();

/*      To read temperature before collect smoke data   */
    temp[0][0] = atof(strtok(scasnd(),"/"));
    for(i=1;i<4;i++)
        temp[0][i] = atof(strtok(NULL,"/"));
/****** To program the 8253 timer chip                              */
    tm_set(NEWLAT); /* send 100 pulses/sec */
    gotoxy(10,4);
    cputs("Timer chip (8253) is programmed !");

/****** To transmit the input and receive the output                */
    gotoxy(10,5);
    cputs("Signal generation and data collection are started !");
    gotoxy(13,6);
    ii = NUMSIG*SETDAT*SMPTM/6000+1.0;
    cprintf("Wait about %4.1f minutes ..... ",ii);
    gotoxy(13,7);
/*    cputs(" Input        Smoke (%)");  */
    biostime(1,0);
```

```
    for ( i=0 ; i<SETDAT ; i++ )
    {
        for ( j=0 ; j<NUMSIG ; j++ )
        {
            if(signal[j]==1)                /* send input signal */
                setsnd();
            else
                clesnd();
            rawdata[i][j] = atof(chasnd());/* receive output */
/*            gotoxy(14,8);
            cprintf("%2d           %8.5f",signal[j],rawdata[i][j]);*/
            while((biostime(0,0))%SMPTM) k=1;   /* time delay */
        }
     }
/****** To restore the 8253 timer chip                                  */
    tm_set(OLDLAT);
    gotoxy(10,7);
    cputs("Timer chip is restored !");
/*      To read temperature after collect smoke data   */
    temp[1][0] = atof(strtok(scasnd(),"/"));
    for(i=1;i<4;i++)
        temp[1][i] = atof(strtok(NULL,"/"));

/****** To record data on the file                                      */
    gotoxy(10,8);
    cputs("Data are being recorded on the floppy....");
    fprintf(fo,"\tTemp.(deg. C)  Before  After\n");
    fprintf(fo,"\tEngine oil  : %5.1f  %5.1f\n",temp[0][0],temp[1][0]);
    fprintf(fo,"\tCoolant     : %5.1f  %5.1f\n",temp[0][1],temp[1][1]);
    fprintf(fo,"\tInlet air   : %5.1f  %5.1f\n",temp[0][2],temp[1][2]);
    fprintf(fo,"\tExhaust gas : %5.1f  %5.1f\n",temp[0][3],temp[1][3]);
        fprintf(fo,"\n\t  No. Input  Smoke (%)\n");
    for ( i=0 ; i<NUMSIG ; i++ )
    {
        data[i] = 0.0;
        for( j=1 ; j<SETDAT ;j++ )
            data[i] += rawdata[j][i];
        data[i] /= (SETDAT-1)/100.;
        fprintf(fo,"\t%4d   %2d   %10.5f\n",i,signal[i],
                data[i]);
    }
    gotoxy(10, 9);
    cputs("Data logging is successufully done !");
    fclose(fo);
    exit(0);
}
/* to convert the index signal table to individual signals     */
void con_sgn(int index[])
{
    int i,ii,k,sign;
```

```
    ii=0;
    k=0;
    sign=1;
    for(i=0;i<NUMSIG;i++)
        {
        if(k<index[ii]){
            signal[i]=sign;
            k++;
            }
        else{
            k=0;
            ii++;
            sign=sign*(-1);
            i--;
            }
        }
}
/*****     LOG.C     ************************************************/


;*****     TMCM.ASM     ******************************************
;
;   1. Program name
;      TMCM.ASM
;
;   2. Date
;      April 20, 1988
;      August 21, 1988 (Revised)
;      September 21, 1988 (Revised)
;
;   3. Object
;      1) To program the 8253 timer chip for an accurate
;         control of the real time
;      2) To initialize the serial prot (COM1) for the
;         communication with the Micro-mac
;      3) To send micro-mac commands, receive data and
;         return the sorted data for temperature reading
;         (Added in August)
;
;   4. Usage
;      Subprogram for "LOG.C" and "MENU.C" with
;      Turbo C interface
;
;-----------------------------------------------------------------------
        name    timecomm
_TEXT   segment byte public 'CODE'
DGROUP  group   _BSS,_DATA
        assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT   ends
_DATA   segment word public 'DATA'
```

```
_d@     label   byte
_DATA   ends
_BSS    segment word public 'BSS'
_b@     label   byte
_BSS    ends
_TEXT   segment byte public 'CODE'
;---------------------------------------------------------------------
; To program the timer chip for precise control
;---------------------------------------------------------------------
_tm_set proc    near
        push    si
        push    bp
        mov     bp,sp
        mov     al,36h              ;set up I/O register
        out     43h,al
        mov     ax,word ptr[bp+6]   ;pass counter number
        out     40h,al              ;send counter to latch
        mov     al,ah
        out     40h,al
        pop     bp
        pop     si
        ret
_tm_set endp
;---------------------------------------------------------------------
; To initialize serial port for communication with micro-mac 4000
;---------------------------------------------------------------------
_inicom proc    near
        mov     dx,3fbh         ;Initialize the baud rate divisor
        mov     al,80h          ;registors for 9600 bps
        out     dx,al
        mov     dx,3f9h
        mov     al,0
        out     dx,al
        mov     dx,3f8h
        mov     al,0ch
        out     dx,al
        mov     al,1eh          ;Initialize the line control register
        mov     dx,3fbh
        out     dx,al
        mov     dx,3fch         ;Initialize the MODEM control register
        mov     al,03h
        out     dx,al
        mov     dx,3f9h         ;Initialize the interrupt enable
        mov     al,0            ;register
        out     dx,al
        ret
_inicom endp
;---------------------------------------------------------------------
;To send DEG command and receive response for temperature conversion
;---------------------------------------------------------------------
```

```
;_degsnd proc    near
;        lea     bx,deg_cmd      ;Load DEG command
;        call    send            ;Send command
;        call    receive         ;Receive data
;        ret
;_degsnd endp
;-------------------------------------------------------------------
; To send SCA command and receive and sort data for temperature data
;-------------------------------------------------------------------
_scasnd proc    near
        push    si
        push    bp
        mov     bp,sp
        lea     bx,sca_cmd      ;Load SCA command
        call    send            ;Send command
        call    recesca         ;Receive data
        mov     si,offset sca_dat       ;Return
        mov     ax,si                   ;  to C routine
        pop     bp
        pop     si
        ret
_scasnd endp
;-------------------------------------------------------------------
; Subroutines
;-------------------------------------------------------------------
send    proc    near
back:   mov     dx,3fdh         ;Check if ready to transmit
        in      al,dx           ;
        test    al,20h          ;
        jz      back            ;If not wait
        mov     dx,3f8h         ;Transmit the character from
        mov     al,[bx]         ;the memory buffer
        out     dx,al           ;
        cmp     al,13           ;Check if CR transmittted
        je      exit
        inc     bx
        jmp     back
exit:   ret
send    endp
;-------------------------------------------------------------------
receive proc    near
again:  mov     dx,3fdh         ;Check if data received
        in      al,dx           ;
        test    al,01h          ;
        jz      again           ;If not wait
        mov     dx,3f8h         ;Receive dat
        in      al,dx           ;
        and     al,7fh          ;
;       push    ax              ;Print data on screen
;       mov     bx,0            ;
```

```
;       mov     ah,14           ;
;       int     10h             ;
;       pop     ax              ;
        cmp     al,0ah          ;Check if LF received
        jnz     again
        ret
receive endp
;-------------------------------------------------------------------------
rececha proc    near
        xor     si,si           ;Initializing si,bp with
        xor     bp,bp           ;zero
again1: mov     dx,3fdh         ;Check if data received
        in      al,dx           ;
        test    al,01h          ;
        jz      again1          ;If not wait
        mov     dx,3f8h         ;Receive data
        in      al,dx           ;
        and     al,7fh          ;Mark 7th bit
        cmp     al,':'          ;To sort data
        jnz     skip1           ;
        inc     bp              ;
        jmp     again1          ;
skip1:  cmp     bp,1            ;
        jnz     skip2           ;
        mov     cha_dat[si],al  ;To store at memory buffer
        inc     si
skip2:  cmp     al,0ah          ;Check if LF received
        jnz     again1
        ret
rececha endp
;-------------------------------------------------------------------------
recesca proc    near
        xor     si,si           ;Initializing si,bp with
        xor     bp,bp           ;zero
repeat: mov     dx,3fdh         ;Check if data received
        in      al,dx           ;
        test    al,01h          ;
        jz      repeat          ;If not wait
        mov     dx,3f8h         ;Receive data
        in      al,dx           ;
        and     al,7fh          ;Mark 7th bit
        cmp     al,':'          ;To sort data
        jnz     next1           ;
        inc     bp              ;
        jmp     repeat          ;
next1:  cmp     bp,1            ;
        jnz     next2           ;
        mov     sca_dat[si],al  ;To store at memory buffer
        inc     si
next2:  cmp     al,0ah          ;Check if LF received
```

```
        jnz     repeat
        ret
recesca endp
;-------------------------------------------------------------------------
_TEXT   ends
_DATA   segment word public 'DATA'
_s@     label   byte
cha_dat db      8 dup (?)
sca_dat db      32 dup (?)
set_cmd db      "*0:SET/1/7:",13
cle_cmd db      "*0:CLE/1/7:",13
cha_cmd db      "*0:CHA/0:",13
deg_cmd db      "*1:DEG/C:",13
sca_cmd db      "*0:SCA/8/11:",13
_DATA   ends
_TEXT   segment byte public 'CODE'
        public  _inicom,_scasnd,_tm_set
_TEXT   ends
        end

;*****     TMCM.ASM     ********************************************

/*****     CGRAPHIO.C     ******************************************

    1. Program name
       CGRAPHIO.C

    2. Date
       June 28, 1988

    3. Object
       To draw graphs for input and output in time domain

    4. Usage
       Child program for "MENU.C"

-------------------------------------------------------------------------/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>

#define MGN     0.25            /* graph margin                 */
#define NSMPLS  512             /* number of samples            */

struct PTS {
  int x, y;
```

```
};                      /* Structure to hold vertex points   */

int    GraphDriver;     /* The Graphics device driver*/
int    GraphMode;       /* The Graphics mode value   */
double AspectRatio;     /* Aspect ratio of a pixel on the screen*/
int    maxres[2];       /* The maximum resolution of the screen */
int    MaxColors;       /* The maximum # of colors available    */
int    ErrorCode;       /* Reports any graphics errors          */
struct palettetype palette; /* Used to read palette info    */

float data[2][NSMPLS];

void Initialize(void);
void Do_graph();
void changetextstyle(int font, int direction, int charsize);

main(int argc,char *argv[])
{
    int i;
        FILE *fi;

    if((fi = fopen(argv[1],"r")) == NULL)
        printf("Can't open %s to read.",argv[1]);

    else {
        /*   Read input data and generate points for analysis                  */
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        fscanf(fi,"%*s%*s%*s%*s");
        fscanf(fi,"%*s%*s%*s%*f%*f");
        fscanf(fi,"%*s%*s%*f%*f");
        fscanf(fi,"%*s%*s%*s%*f%*f");
        fscanf(fi,"%*s%*s%*s%*f%*f");
        fscanf(fi,"%*s%*s%*s%*s");
        for(i=0;i<NSMPLS;i++)
            fscanf(fi,"%*d %f %f\n",
                &data[0][i],&data[1][i]);
        Initialize();
        Do_graph();
        closegraph();   /* Return the system to text mode       */
    }
    exit(0);
}

void Initialize(void)
{
    int xasp,yasp;

    GraphDriver = DETECT;     /* Request auto-detection       */
    initgraph( &GraphDriver, &GraphMode, "" );
    ErrorCode = graphresult();/* Read result of initialization*/
```

```
    if( ErrorCode != grOk ){  /* Error occured during init    */
        printf(" Graphics System Error: %s\n",
        grapherrormsg( ErrorCode ) );
        exit( 1 );
    }

    getpalette( &palette );         /* Read the palette from board  */
    MaxColors = getmaxcolor() + 1;  /* Read maximum number of colors*/

    maxres[0] = getmaxx();
    maxres[1] = getmaxy();          /* Read size of screen          */

    getaspectratio( &xasp, &yasp ); /* read the hardware aspect     */
    AspectRatio = (double)xasp / (double)yasp;
                                    /* Get correction factor        */
}

void Do_graph()
{
    int h,w,ci,co,ix,yhalf;
    double min,max,xx,yi,yo,value,check;
    double rng,range[2],base,mul[3];
    int mar[2];
    int i,j,k,xxx,yii,yoo;
    double end[2],step[2],step_num,ini_step_len,step_size[2],start[2];
    char xind[],yind[];
    struct viewporttype vp;

/*  to put title name ,axis names and lable names       */
/*    setcolor(LIGHTCYAN); */
    getviewsettings( &vp );
    changetextstyle(SMALL_FONT, HORIZ_DIR, 7);
    settextjustify( CENTER_TEXT, CENTER_TEXT);
    h = vp.bottom - vp.top;
    w = vp.right - vp.left;
    outtextxy(w/2,0.05*h,"INPUT AND OUTPUT DATA IN TIME DOMAIN");
    changetextstyle(SMALL_FONT,VERT_DIR,6);
    outtextxy(0.04*w,0.33*h,"Fuel");
    outtextxy(0.05*w,0.70*h,"Smoke");
    outtextxy(0.07*w,0.70*h,"(%)");
    outtextxy(0.90*w,0.33*h,"Input");
    outtextxy(0.90*w,0.68*h,"Output");
    changetextstyle(SMALL_FONT, HORIZ_DIR, 6);
    outtextxy(0.6*w,0.96*h,"Time (* 70 mSec)");
    yhalf = maxres[1]/2;
    mar[0] = maxres[0]/7;
    mar[1] = maxres[1]/6;
    /* to draw border box */
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    rectangle(mar[0],mar[1],maxres[0]-mar[0],yhalf-3);
```

```
    rectangle(mar[0],yhalf+3,maxres[0]-mar[0],maxres[1]-mar[1]);

/*  to put scales on each axis          */
    for(j=0;j<2;j++){
        min = 100.0;
        max = -100.0;
        for(i=0;i<NSMPLS;i++){
            if(max < data[j][i])
                max = data[j][i];
            if(min > data[j][i])
                min = data[j][i];
        }
        rng = fabs(max - min);
        base = min - MGN * rng;
        /* to determine graphic range of data */
        ini_step_len = pow10(floor(log10(rng)));
        step_num = floor(rng/ini_step_len);
        if(step_num > 10.0)
            step[j] = ini_step_len*2.0;
        else if(step_num <= 10.0 & step_num > 5.0)
            step[j] = ini_step_len;
        else if(step_num <=2.0 & step_num > 1.0)
            step[j] = ini_step_len/5.0;
        else if(step_num <=1.0)
            step[j] = ini_step_len/10.0;
        else
            step[j] = ini_step_len/2.0;
        value = fabs(fmod(base,step[j]));
        check = step[j]/10.0;
        if(fabs(value) > check & fabs(value) < (step[j]-check)){
            if(base>=0)
                start[j] = base - value;
            else
                start[j] = base + value -step[j];
        }
        else
            start[j] = base;

        /* to find a actual interval to draw */
        end[j] = start[j];
        while(end[j] < (max-step[j]/2.0+rng*MGN))
            end[j] += step[j];
        range[j] = end[j] - start[j];
    }
    /* to calculate the multiplcating factors to plot */
    mul[0] = (yhalf-3-mar[1])/range[0];
    mul[1] = (maxres[1] - yhalf -3 -mar[1])/range[1];
    /* to draw scale on each axix */
    for(j=0;j<2;j++)
        step_size[j] = step[j]*mul[j];
```

```
    changetextstyle(SMALL_FONT, HORIZ_DIR,4);
    for(i=0,yi=start[0];yi<=end[0];i++,yi+=step[0]){
        yii =mar[1]+ (end[0] - yi) * mul[0];
        moveto(mar[0]-4,yii);
        linerel(4,0);
        if(!(i%3))
            outtextxy(mar[0]-20,yii,gcvt(yi,2,xind));
        }
    for(i=0,yo=start[1];yo<=end[1];i++,yo+=step[1]){
        yoo =yhalf+3+ (end[1]-yo) * mul[1];
        moveto(mar[0]-4,yoo);
        linerel(4,0);
        if(!(i%3))
            outtextxy(mar[0]-20,yoo,gcvt(yo,2,xind));
    }
    for(i=0;i<=5;i++){
        xxx = mar[0] + i*100.0 * (maxres[0] - 2*mar[0])/512;
        moveto(xxx,maxres[1]-mar[1]);
        linerel(0,4);
        outtextxy(xxx,maxres[1]-mar[1]+15,gcvt(i*100.0,2,xind));
    }
    /* to convert data point to pixel location */
/*    setcolor(YELLOW); */
    for(i=0;i<NSMPLS;i++) {
        ci =mar[1]+ (end[0] - data[0][i]) * mul[0];
        ix =mar[0] + 1.0*i*(maxres[0] - 2*mar[0])/NSMPLS;
        if(i)
            lineto(ix,ci);
        moveto(ix,ci);
    }
    for(i=0;i<NSMPLS;i++) {
        co =yhalf+3+ (end[1] - data[1][i]) * mul[1];
        ix =mar[0] + 1.0*i*(maxres[0] - 2*mar[0])/NSMPLS;
        if(i)
            lineto(ix,co);
        moveto(ix,co);
    }
    getch();
    closegraph();
    cleardevice();
}

void changetextstyle(int font, int direction, int charsize)
{
    int ErrorCode;

    graphresult();                          /* clear error code       */
    settextstyle(font, direction, charsize);
    ErrorCode = graphresult();              /* check result           */
    if( ErrorCode != grOk ){                /* if error occured       */
```

```
        closegraph();
        printf(" Graphics System Error: %s\n",
            grapherrormsg( ErrorCode ) );
        exit( 1 );
    }
}
/*****     CGRAPHIO.C     ********************************************/
```

APPENDIX G: PROGRAM LISTING FOR DATA ANALYSIS

```
/*****     FRQPRBS.C     ********************************************

    1. Program name
       FRQPRBS.C

    2. Date
       October 1988

    3. Object
       To calculate system frequency response from PRBS
       data collected using power spectrum density

-------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>

#define     PI      3.141592653589795
#define     NSMPLS  512             /* Number of samples           */
#define     N2      9               /* Power of 2 of total signals */
#define     T       0.07            /* Sampling time in sec.       */
#define     SMALL   0.00001         /* Small value for  calculation */
#define     NFREQ   NSMPLS/2

void trnsfr();                      /* Calculating transfer function */
int input[NSMPLS];
float  output[NSMPLS];
    /* Signals of I/O in time domain */
double  magtrn[NFREQ],angtrn[NFREQ],w[NFREQ];
     /*   magtrn      ; Magnitude of transfer function
          angtrn      ; Phase angle of transfer function
          w       ; Frequency, rad/min                          */

/*-------------------------------------------------------------------*/
main()
{
    int i;
    char filein[13],fileout[13];
    FILE *fi,*fo;

printf("Enter the data file name = ");
scanf("%s",filein);
    printf("Enter the output file name = ");
scanf("%s",fileout);
    if((fi = fopen(filein,"r")) == NULL){
```

```
            printf("Can't open %s input file.\n");
    exit(0);
        }

        fo = fopen(fileout,"w");
        /*  read input data     */
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        for(i=0;i<NSMPLS;i++)
            fscanf(fi,"%*d %d %f\n",&input[i],&output[i]);

        /*  call subroutine to calculate the transfer function      */
        trnsfr();
        /*  print results    */
        fprintf(fo,"\n\n\t\t\tFrequency Response of System\n\n");
fprintf(fo,"\t No.  Omega(rad/s) Log10(omg)");
fprintf(fo,"    Amp.      Phase \n");
        for(i=1;i<NFREQ;i++){
            fprintf(fo,"\t%3d %10.6f  %10.6f  %10.6f  %10.6f\n",
            i,w[i],log10(w[i]),magtrn[i],angtrn[i]);
        }
        /*  closing the program     */
        fclose(fi);
        fclose(fo);
 }
/*--------------------------------------------------------------------*/
/*function for calculating system transfer function                   */
void trnsfr()
{
    int i,j,k;
    struct complex spec[NSMPLS][2];
    struct complex cmpin[NSMPLS],cmpout[NSMPLS];
    double  magntd[2],angle[2];
     /* i  ; Sample order index
       j  ; I/O index, J=1; Input, J=2; Output
        spec(I,J)    ; Power spectrum of i/o
        magntd(J)    ; Magnitude of frequency domain signal
        angle(J)     ; Angle of frequency domain signal   */

/*   Calculate power spectrum of input/output signals   */

        for(i=0;i<NSMPLS;i++){
            cmpin[i].x=input[i]*1.;
            cmpout[i].x = output[i];
            cmpin[i].y=0.;
            cmpout[i].y = 0.;
        }
        fft(cmpin,N2,-1);
```

```
fft(cmpout,N2,-1);

        for(i=0;i<NSMPLS;i++) {
    spec[i][0].x=cmpin[i].x*cmpin[i].x+cmpin[i].y*cmpin[i].y;
    spec[i][0].y=cmpin[i].x*cmpin[i].x*cmpin[i].y-cmpin[i].y*cmpin[i].x;
    spec[i][1].x=cmpin[i].x*cmpout[i].x+cmpin[i].y*cmpout[i].y;
    spec[i][1].y=cmpin[i].x*cmpout[i].y-cmpin[i].y*cmpout[i].x;
        }

/*   Finding the transfer function    */

    for(i=0;i<NFREQ;i++){
for(j=0;j<2;j++) {

/*    Calculating magnitude of I/O      */
          magntd[j] = cabs(spec[i][j]);

/*    Calculating phase angle of I/O     */
          if(fabs(spec[i][j].x) <= SMALL){
    if (spec[i][j].y == 0.)
    angle[j] = 0.;
              else if (spec[i][j].y < 0.)
                  angle[j] = -PI/2.;
              else
                  angle[j] = PI/2.;
          }
          else
              angle[j] = atan2(spec[i][j].y,spec[i][j].x);
        }

/*    Calculating system transfer function
      by magnitude and phase angle          */
        w[i] = 2.*PI*(i)/NSMPLS/T;
        if(magntd[0] <= SMALL){
              printf("Magnitude became zero at %3dth sigal.\n",i);
magtrn[i] = 0.;
 }
else
        magtrn[i] = magntd[1]/magntd[0];
        angtrn[i] = angle[1]-angle[0];
        if(angtrn[i] > 0.)
            angtrn[i] = angtrn[i] - 2.* PI;
    }
}
/*****     FRQPRBS.C     ********************************************/


/*****     FRQMFBS.C     ********************************************

    1. Program name
```

```
      FRQMFBS.C

   2. Date
      June 1988

   3. Object
      To calculate system frequency response from MFBS
      data collected using fast Fourier transform

-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "conio.h"

#define     PI      3.141592653589795
#define     NSMPLS  512              /* Number of samples            */
#define     N2      9               /* Power of 2 of total signals   */
#define     NFREQ   8                /* Number of frequencies        */
#define     SMALL   0.00001          /* Small value for  calculation */

void trnsfr();                       /* Calculating transfer function */
struct  complex cmpsig[NSMPLS];
     /*   cmpsig(I)   ; FFT variables                                  */
float  tmsig[NSMPLS][2];        /* Signals of I/O in time domain     */
float T; /* sampling time,sec                    */
double  magtrn[NFREQ],angtrn[NFREQ],w[NFREQ],w[NFREQ];
     /*   magtrn      ; Magnitude of transfer function
          angtrn      ; Phase angle of transfer function
          w         ; Frequency, rad/min                          */
int     vfreq[NFREQ] = {1,2,3,4,6,8,12,16};
     /*   vfreq(K)    ; Values of Frequency studied                 */

/*-------------------------------------------------------------------------*/
main()
{
    int i;
    char filein[13],fileout[13];
    FILE *fi,*fo;

        printf("Enter the data file name = ");
        scanf("%s,filein);
        printf("Enter the output file name = ");
        scanf("%s",fileout);
        T = 0.07;
        if((fi = fopen(filein,"r")) == NULL){
            printf("Can't open %s input file.\n");
            exit(0);
        }
```

```
        fo = fopen(argv[2],"w");
        /*  read input data     */
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        for(i=0;i<NSMPLS;i++)
            fscanf(fi,"%*d %f %f\n",&tmsig[i][0],&tmsig[i][1]);
        /*  call subroutine to calculate the transfer function      */
        trnsfr();
        /*  print results    */
        fprintf(fo,"\n\n\t\tFrequency Response of System\n\n");
        fprintf(fo,
           "\tOMEGA(rad/m)\t   AMP. RATIO \t\t  PHASE ANGLE(rad)\n\n");
        for(i=0;i<NFREQ;i++){
            fprintf(fo,"\t%10.8f\t%18.16f \t%19.16f\n",
            w[i],magtrn[i],angtrn[i]);
        }
        /*  closing the program     */
        fclose(fi);
        fclose(fo);
    }
    exit(0);
}
/*-------------------------------------------------------------------------*/
/*function for calculating system transfer function                        */
void trnsfr()
{
    int i,j,k;
    /* i  ; Sample order index
       j  ; I/O index, J=1; Input, J=2; Output
       k  ; Frequency order index                                    */
    struct  complex fourie[NSMPLS][2],fouomg[2];
    /* fourie(I,J) ; Fourier transformed signal
       fouomg(J)   ; Fourier transformed signal at certain frequency */
    double  magntd[2],angle[2],a[2],b[2];
    /* magntd(J)   ; Magnitude of frequency domain signal
       angle(J)    ; Angle of frequency domain signal
       a(J)        ; Real part of frequency domain signal
       b(J)        ; Imaginary part of frequency domain signal  */

/*   Fast Fourier transform of input/output signals   */

    for(j=0;j<2;j++){
        for(i=0;i<NSMPLS;i++){
            cmpsig[i].x=tmsig[i][j];
            cmpsig[i].y=0.;
        }
        fft(cmpsig,N2,-1);
        for(i=0;i<NSMPLS;i++){
            fourie[i][j].x=cmpsig[i].x;
            fourie[i][j].y=cmpsig[i].y;
        }
```

```
    }

/*   Finding the transfer function    */

    for(i=0;i<NFREQ;i++){
        k = vfreq[i];
        for(j=0;j<2;j++){

/*    Choosing transformed signal at certain frequency    */
            fouomg[j].x = fourie[k][j].x;
            fouomg[j].y = fourie[k][j].y;

/*    Calculating magnitude of I/O      */
            magntd[j] = cabs(fouomg[j]);
            if(magntd[j] <= SMALL)
                printf("Magnitude is zero !!!\n");

/*    Calculating phase angle of I/O     */
            else{
                a[j] = fouomg[j].x;
                b[j] = fouomg[j].y;
            }
            if(fabs(a[j]) <= SMALL){
                if(fabs(b[j]) < 0.)
                    angle[j] = -PI/2.;
                else
                    angle[j] = PI/2.;
            }
            else
                angle[j] = atan2(b[j],a[j]);
        }

/*    Calculating system transfer function
      by magnitude and phase angle          */

        w[i] = 2.*PI*60.*vfreq[i]/NSMPLS/T;
        magtrn[i] = magntd[1]/magntd[0];
        angtrn[i] = angle[1]-angle[0];
        if(angtrn[i] > 0.)
            angtrn[i] = angtrn[i] - 2.* PI;
    }
}
/*****     FRQMFBS.C     ********************************************/


/*****     CURP.C     ************************************************

    1. Program name
       CURP.C
```

```
    2. Date
       June 1988

    3. Object
       To search the transfer function coefficients from
       frequency response result using direct search method.

    4. Usage
       For PRBS data

------------------------------------------------------------------------
     Main variable listing

        NSMPLS      ; Number of samples
        NUMX        ; Number of explicit independent variables
        ALPHA       ; Reflection factor
        BETA,GAMMA  ; Convergence parameter
        DELTA       ; Explicit constraint violation correction
        xvar        ; Independent variables
        ran_one     ; Random numbers between 0 and 1
        objfun      ; Objective function
        ITMAX       ; Maximum number of iterations
        it          ; Iteration index
        locons      ; Lower constraints
        hicons      ; Higher constraints
        magtrn      ; Magnitude of transfer function
        angtrn      ; Phase angle of transfer function
        w           ; Frequency, rad/min
        centrd      ; Centroid
        imax        ; index of point with maximum function value
        imin        ; index of point with minimum function value

------------------------------------------------------------------------*/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

#define     NUMSET  10          /* Number of points in the complex */
#define     NUMX    7           /* Number of coef. on the T(s) */
#define     NFREQ   36          /* Number of frequencies  */
#define     ALPHA   1.3/* Parameter value of search */
#define     BETA    0.001 /*"*/
#define     GAMMA   30/*"*/
#define     DELTA   0.0001/*"*/
#define     ITMAX   5000/* Maximum number of iteration */
#define     MAXCOEF 1./* Limit of search for gain*/
#define     TRUE    1
```

```
void searching();
void pre_complex();
void check(int i);
void cent_roid(int imax);
void obj_func(int itry);
void ran_dom();

double xvar[NUMSET][NUMX],objfun[NUMSET];
float ran_one[NUMSET][NUMX],centrd[NUMX],locons[NUMX];
float hicons[NUMX],w[NFREQ];
struct complex trndat[NFREQ];
int dead,order,imax,imin,it,count;
float beta;

main(argc,argv)
int argc;
char *argv[];
{
    int i,j;
    float magtrn[NFREQ],angtrn[NFREQ],w[NFREQ];
    char filein[20],fileout[20];
    FILE *fi,*fo;

/*printf("Enter the file name for input : ");
scanf("%s",filein);
printf("Enter the file name for output : ");
scanf("%s",fileout);
        if((fi = fopen(filein,"r")) == NULL) {
            printf("Can't open %s to read.",filein);
    exit(0);
        }
        fo = fopen(fileout,"w"); */
        if((fi = fopen(argv[1],"r")) == NULL) {
            printf("Can't open %s to read.",argv[1]);
    exit(0);
        }
        fo = fopen(argv[2],"a");

        /*   Read input data for analysis                */
        fscanf(fi,"%*s%*s%*s%*s%*s%*s%*s%*s%*s");
        for(i=0;i<NFREQ;i++)
            fscanf(fi,"%*d %f %*f %f %f\n",
                &w[i],&magtrn[i],&angtrn[i]);
        printf("Searching coefficients by direct search method ...\n");
/*        for(i =0;i<5;i++)
            printf("%f %f %f\n",w[i],magtrn[i],angtrn[i]);      */
        /*  initialization  */
        for(j=0;j<NUMX;j++)
            xvar[0][j] = 0.;
        /*  calculating transfer function at each frequency  */
```

```
        for(i=0;i<NFREQ;i++){
            trndat[i].x = magtrn[i]*cos(angtrn[i]);
            trndat[i].y = magtrn[i]*sin(angtrn[i]);
/*    printf(" %2d trndat = %f %f\n",i,trndat[i]); */
        }
beta = BETA;
        /* optimizing iterations while modifying the beta values */
        while(TRUE)
        {
        /*  call subroutine to calculate the transfer function */
            it=0;
            searching();
            /*  print results    */
            if(it >= ITMAX){
                beta += BETA;
                printf("Beta value is modified to %5.3e\n",beta);
            }
            else {
                fprintf(fo,"\n\n\t\tModel Parameters of System\n\n");
                for(i=0;i<NUMX;i++)
                    fprintf(fo,"%8.4f ",xvar[imin][i]);
                fprintf(fo,"\n");
                fprintf(fo,"Objfunc = %f\n",objfun[imin]);
                break;
           }
        }

        /*   Closing the function     */
        fclose(fi);
        fclose(fo);
}
/*-------------------------------------------------------------------*/
/* function for searching */
void searching()
{
    static float ff = 10000.;
    static int inloop = 0;
    int i,ii,j,jj,icheck,loop;
    double diff;

/*  set the constraints limit */
    for(i=0;i<NUMX; i++) {
        locons[i] = 0.;
hicons[i] = MAXCOEF;
    }

/*
  prepare starting complex points */
    pre_complex();
```

```
/*  evaluate the object function at each set of values */
    for(i=0;i<NUMSET;i++)
        obj_func(i);

/*  iterate itmax times or smaller to find parameters */
    while(ITMAX-it)
    {

/*  find point with highest function value  */
        imax = 0;
        for(i=1;i<NUMSET;i++){
            if(objfun[imax] < objfun[i])
                imax = i;
        }

/*  find point with lowest function value */
        imin = 0;
        for(i=1;i<NUMSET;i++){
            if(objfun[imin] >= objfun[i])
                imin = i;
        }


/*  check convergence criteria */
        diff = objfun[imax]-objfun[imin];
        if(diff <= beta){
            count++;
            if(count >= GAMMA){
                if(ff <= objfun[imin]){
                    inloop++;
                    if(inloop > GAMMA)
                        return;
                }
                else
                    ff = objfun[imin];
            }
        }
        else
            count = 0;
        if(objfun[imin] <= beta)
            return;

/*   Replace point with highest function value  */
        cent_roid(imax);
        for(jj=0;jj<NUMX;jj++)
        xvar[imax][jj] = (1.0 + ALPHA)*centrd[jj]-ALPHA*xvar[imax][jj];
        check(imax);
        obj_func(imax);

/*   Replace new points if it repeats as highest function values  */
```

```
        loop = 0;
        icheck = 0;
        for(i=1;i<NUMSET;i++){
            if(objfun[icheck] < objfun[i])
                icheck = i;
        }

/*   Replace new points with random numbers if those become same as
     centroids   */
        if(icheck==imax)
        {
            if(loop >5){
                printf("Started again with new sets of values\n");
                pre_complex();
                for(i=0;i<NUMSET;i++)
                    obj_func(i);
                break;
            }
            else {
                loop++;
                for(jj=0;jj<NUMX;jj++)
                    xvar[imax][jj] = (xvar[imax][jj]+centrd[jj])/2.;
                check(imax);
                obj_func(imax);
                icheck = 0;
                for(i=1;i<NUMSET;i++){
                    if(objfun[icheck] < objfun[i])
                        icheck = i;
                }
            }
        }
        it++;
    }
}

/*-------------------------------------------------------------------*/
/* Calculate complex points and check against constraints */
void pre_complex()
{
    int i,j;

    it=1;
    ran_dom();
    for(i=0;i<NUMSET;i++){
        for(j=0;j<NUMX;j++)
            xvar[i][j] = locons[j]+ran_one[i][j]*(hicons[j]-locons[j]);
        check(i);
    }
    count=0;
}
```

```
/*-------------------------------------------------------------------*/
/*  function for constraints check  */
void check(int i)
{
    int j;

    for(j=0;j<NUMX;j++) {
        if(xvar[i][j] <= locons[j])
            xvar[i][j] = locons[j] + DELTA;
        else if(xvar[i][j] >= hicons[j])
            xvar[i][j] = hicons[j] - DELTA;
        else
            continue;
    }
}

/*-------------------------------------------------------------------*/

/* function for calculating centroids  */
void cent_roid(int imax)
{
    int i,j;

    for(j=0;j<NUMX;j++) {
        centrd[j] = 0.;
        for(i=0;i<NUMSET;i++)
            centrd[j] += xvar[i][j];
        centrd[j] = (centrd[j]-xvar[imax][j])/(NUMSET-1);
    }
}

/*-------------------------------------------------------------------*/
/* function for calculating objective function  */
/*
/* Transfer function, T(s)

          G(aS+1)exp(-DS)
T(s) = ------------------------
       (bS+1)(cS+1)(dS+1)(eS+1)

                 where G =xvar[itry][0]
       a -- e = xvar[itry][1 --- 5]
       D = xvar[itry][6]
                                                              */

void obj_func(int itry)
{
    int i,j,k,ii,iii;
    struct complex trncal,err;
```

```
    double temp,temp2,temp3,cosdw,sindw;

        objfun[itry]=0.;
for(j=0;j<NFREQ;j++){
            /* Calculating denominator */
    trncal.x =1.-xvar[itry][2]*xvar[itry][3]*w[j]*w[j];
    trncal.y = (xvar[itry][2]+xvar[itry][3])*w[j];

            temp = trncal.x;
    trncal.x -= trncal.y*xvar[itry][4]*w[j];
    trncal.y += temp*w[j]*xvar[itry][4];

            temp = trncal.x;
    trncal.x -= trncal.y*xvar[itry][5]*w[j];
    trncal.y += temp*w[j]*xvar[itry][5];

    /* Inverse of denominator */
    temp = trncal.x*trncal.x+trncal.y*trncal.y;
    trncal.x /= temp ;
    trncal.y /= -temp;

    /* Multiply by gain */
    trncal.x *= xvar[itry][0];
    trncal.y *= xvar[itry][0];

    /* Multiply by numerator */
    temp = trncal.x;
    trncal.x -= xvar[itry][1]*trncal.y*w[j];
    trncal.y += temp*w[j]*xvar[itry][1];

    /* Multiply by dead time term */
    temp2 = trncal.x;
    temp3 = xvar[itry][6]*w[j];
    cosdw = cos(temp3);
    sindw = sin(temp3);
    trncal.x = trncal.x*cosdw + trncal.y*sindw;
    trncal.y = trncal.y*cosdw - temp2*sindw;

            err.x=(trndat[j].x-trncal.x);
            err.y=(trndat[j].y-trncal.y);

            objfun[itry] += err.x*err.x+err.y*err.y;
        }
}
/*-------------------------------------------------------------------*/
/*  random number generation */
void ran_dom()
{
    int i,j;
```

```
    randomize();
    for(i=1;i<NUMSET;i++){
        for(j=0;j<NUMX;j++)
            ran_one[i][j] = rand()/32767.;
    }
}
/*****     CURP.C     ***********************************************/


/*****     BODE.C     ***********************************************

    1. Program name
       BODE.C

    2. Date
       October 1988

    3. Object
       To find the magnitude and phase angle for bode plot from
       the Laplace function [ exp(-Ds)( a*S¢2 + b*S + c ) ¢flag ]

    4. Usage
       Used to obtain the frequency response from known parameters

------------------------------------------------------------------------/

#include <alloc.h>
#include <math.h>
#include <stdio.h>

#definePI3.141592653589795
main()
{
    int i,j,neq,flag,n,nf;
    float a,b,c,smpltm,dead;
    double *w,*totmag,*phase,angle,temp,temp2,temp3,cosdw,sindw;
    struct complex resp;
    char filein[13];
FILE *fo;
/*      printf("Enter the number of sample (int) : ");
    scanf("%d",&n);
        printf("Enter the sampling time in sec (float) : ");
        scanf("%f",&smpltm);  */
        n = 512;
        smpltm = 0.07;
        nf = n/2;

        w = (double *) calloc(nf,sizeof(double));
        totmag = (double *) calloc(nf,sizeof(double));
        phase = (double *) calloc(nf,sizeof(double));
```

```
        for(i=0;i<nf;i++) {
            *(w+i) = 2.*PI*i/(n*smpltm);
            *(totmag+i) = 1.;
            *(phase+i) = 0.;
}
printf("\n\tLaplace function form is the multiple of\n");
printf("\n\tequation [ ( A*S*S + B*S + C ) ** Flag ]\n\n");
        printf("Enter the number of equations (int) : ");
scanf("%d",&neq);


        for(j=0;j<neq;j++) {
    printf("Enter the A, B, C, dead,(float) and Flag (int) : ");
    scanf("%f %f %f %f %d",&a,&b,&c,&dead,&flag);
    for(i=0;i<nf;i++) {
        resp.x = c - a * (*(w+i)) * (*(w+i)) ;
    resp.y = b * (*(w+i));
        if(flag == -1) {
    resp.x /= (temp=resp.x*resp.x+resp.y*resp.y);
        resp.y /= -temp;
if(j==0 && dead!=0) {
    temp2 = resp.x;
            temp3 = dead*(*(w+i));
    cosdw = cos(temp3);
            sindw = sin(temp3);
    resp.x = resp.x*cosdw + resp.x*sindw;
                    resp.y = temp2*cosdw - temp2*sindw;
}

            }
            *(totmag+i) *= cabs(resp);
            if(fabs(resp.x))
angle = atan2(resp.y,resp.x);
        else {
        if(!fabs(resp.y))
            angle = 0.;

else if(resp.y > 0)
angle = PI/2.;
else
angle = -PI/2.;
                }
        *(phase+i) += angle;
            }
}
/*        printf("\n\tOmega(rad/s) Log\t Magnitude\t Phase angle\n");
        for(i=1;i<nf;i++) {
            printf("\t%10.6f\t%10.6f\t%10.6f\n",
    *(w+i),*(totmag+i),*(phase+i));
```

```
}
*/

/*printf("Do you want to print the results on the file (y/n)? ");
    if((getche()) != 'y' && (getche()) != 'Y' ) exit(0); */
printf("\nEnter the file name : ");
scanf("%s",filein);
    if((fo = fopen(filein,"w")) != NULL) {
    fprintf(fo,
    "\tOmega(rad/s)\t Magnitude\t Phase angle\tDB\tlog(w)\n");
    for(i=1;i<nf;i++)
        fprintf(fo,"\t%10.6f\t%10.6f\t%10.6f\t%10.6f\n",
    *(w+i),log10(*(w+i)),*(totmag+i),
    *(phase+i));
fclose(fo);
}
}
/*****     BODE.C     ************************************************/
```